TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SERGIO ROMERO SALAS

MAPPING OF A PARTIALLY KNOWN AREA USING COLLABORATION OF TWO NAO ROBOTS

Master of Science Thesis

Examiner: prof. Risto Ritala
Examiner and topic approved on 31st May 2017

ABSTRACT

SERGIO ROMERO SALAS: Mapping of a partially known area
using collaboration of two NAO robots
Tampere University of technology
Master of Science Thesis, 49 pages, 18 Appendix pages
September 2017
Master in Industrial Management (Degree at Universidad Carlos III de Madrid)
Major: Industrial Management
Examiner: Professor Risto Ritala

Keywords: NAO Robot, mapping, landmark, C++, NAOqi

Localization and mapping of unknown areas is a key topic in nowadays humanoid robotics and all the application and services that this field of engineering may have in the future. Furthermore, collaboration always makes tasks easier and improve the productivity of processes.

Taking this into account, the aim of this master thesis is to develop an algorithm to make two humanoid robots work together to realize the mapping of a partially known area. For that purpose, NAO humanoid robots have been used and some landmarks that those robots are able to recognize. The location of the robots is acquired based on the relative position of some common landmarks whose situation in the environment is previously known and all the robots can detect. After the localization of the robots, they will locate other objects in the map, represented with landmarks as well, using the relative position with that landmarks and their own position in the map. Thus, all robots will have the data of the location of all the robots and landmarks without the need of seeing them, using the information sharing. The code has been implemented in the programming language C++ and using the NAOqi API.

After the implementation of the program, some experiments have been carried out with two NAO robots in an indoor environment to measure the accuracy of the location of the robots as well as the unknown landmarks in the map. It has been studied the effect of some drivers as the number of known landmarks, the distance between them or the speed of the head movement in the accuracy and reliability of the mapping task.

PREFACE

I would like to acknowledge the department of Automation and Hydraulic for providing me all the equipment for the development of this Master Thesis, as well as all the people in the department for their help and kindness. I want to thank especially Risto Ritala for all the support received from him not just in the academical part, but also in all the processes linked with the thesis on the bureaucratic part.

My gratitude is also for all people I've met in Tampere University of Technology and for the country of Finland, which made me feel like home on this year I spent there and turn this course, without any doubt, into one of the best years of my life.

Copenhague, 16.09.2017

Sergio Romero Salas

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

API	Application Program Interface
AUV	Autonomous Underwater Vehicle
csv	Comma Separated Values
IDE	Integrated Development Environment
IP	Internet Protocol
OS	Operative System
QML	Qt Meta Language
QVGA	Quarter Video Graphics Array
SDK	Software Development Kit
SLAM	Simultaneous Localization and Mapping
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus

%	percent
α	angle between frames
^{o}	degrees
μm	micrometers
a, b	sin(α) and cos(α)
cm	centimeters
dB	decibels
m	meters
rad	radians
sec	seconds
V	volts
Wh	*watt-hour*
x_{global}, y_{global}	X and Y coordinates in global frame
x_n, y_n	X and Y coordinates in NAO local frame
x_o, y_o	X and Y coordinates of NAO in the global frame

1 INTRODUCTION

In this chapter, firstly the motivation of the project will be described, followed by the presentation of the objectives, the state of the art and finally the structure of the whole document will be presented as well.

1.1 Motivation

Despite the voice of some experts that argue that technological progress may destroy humankind in the future [1], the progress of humankind and society has always been directly linked with the development of the technology. We can find an example of this relation in the industrial revolution. Before this technological revolution, most of the standards of living and welfare were much lower than afterwards, as we can see in the Figure 1.1, mainly because of very hard manual tasks were no longer carried out by humans, but performed by machines.

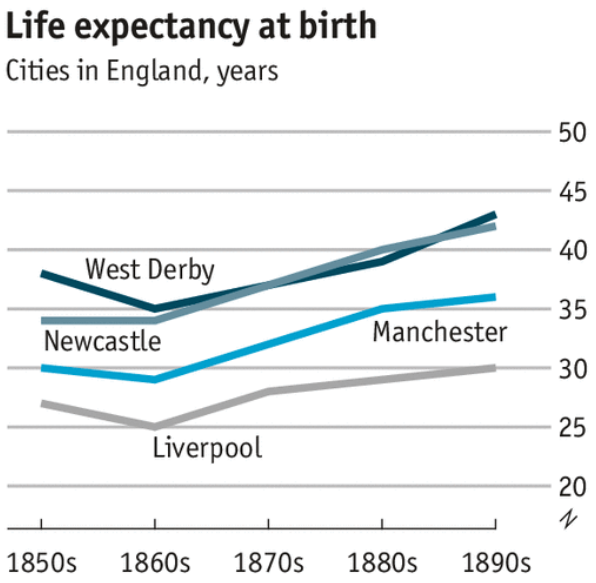

Figure 1.1 *Evolution of life expectancy during the years of industrial revolution in some cities of Great Britain.* ***[2]***

Nowadays, a key field in this technological development is the automation of productive processes and services, which is one of the topics where companies and educational institutions are putting more effort in order to improve business benefit and social welfare. Some experts forecast a new industrial revolution in the next years, where the humans will not be longer needed in productive processes.

It is clear that in all the automation and robotic task we have to handle nowadays, localization and mapping of environments is an essential feature to be improved and optimized. This importance is obviously bigger in the domain of humanoid and service robotics, where the localization of the robot in the environment it's working in, as well as the elements necessary for its work, is crucial for the correct realization of its task.

Mapping and localization is an applicable technique in a wide range of different domains. Between them, we can find the domestic robots, as cleaning floor robots, where the locations of themselves and the limits of the house are very important, or robots for helping disability and/or older people, turning the task of localization of the human and objects into a crucial issue.

Outside home, the task of mapping is also found important in other services as it could be waiter, drones for security and other issues or even in the field of vehicles, with the development observed in the last years from part of some companies working in autonomous vehicles, topic that we'll extend more in the state of the art. The application of localization in robotics is not just limited to daily tasks, but also to the progress of humankind, since it's an essential part of the space exploration, especially in robots sent to other planets.

In addition to all this fields of application of mapping and localization, it's remarkable that, in the next years, new applications will appear with the fast development we will experiment in the major of automation and robotics, so it makes mapping, considering the present utilization as well, a key field in next years.

It is also important to take into account that, as it happens in the human behavior, a cooperation between different individuals always makes tasks realization easier and more efficient. That's also something we should consider in the future of robotics, where several robots will have to cooperate to achieve a goal, and this idea has also been applied in this thesis since the program will be developed for the cooperation in the mapping of an area between several robots.

The implementation of this thesis has been done for inside environments in controlled situations for the robots, but with the idea that this work could be a starting point for future work in real outside environments.

To sum it up, the motivation of this master thesis it to make a contribution to the mapping and localization task present in the nowadays robotic field and with a huge potential in the close future, by making robots collaborate to be able to map a partially known area in a more efficient way, and try to improve people's life making it easier and more safety in the future.

1.2 Objectives of the project

Now, we will continue with the list of the several objectives that covers this master thesis.

- Develop a consistent and reliable algorithm for NAO robots that allow several robots to map a partially-known area by the use of the information of the others provide. This is the main objective of the project.
- Test the code developed in an indoor environment using landmarks that NAO robot can identify. Study different drivers as precision in the mapping, influence of the distance between the robots, mistakes in landmarks recognition, etc.
- Make a contribution to the Automation and Hydraulic Engineering department's research in Tampere University of Technology in the field of autonomous robots' mapping and localization and create a start point with a C++ code for mapping a partially-known area using the collaboration of several robots.

In addition to these objectives, in a more personal level, there were also a few objectives that were achieved during the realization of this thesis.

- Learn about humanoid robots' hardware and software and how to interact with them and program them, particularly NAO robot and all his framework (NAOqi API (Application Programming Interface), C++ SDK (Software Development Kit), etc.), as well as understanding the work with NAO landmarks detection.
- Review and improve the knowledge of C++ programming and debugging obtained during the education before and apply it to a practical case as it can be the localization and mapping using a humanoid robot.

1.3 State of the art

In this section, it will be described the actual state of the art in the field of mapping and localization of robotics.

Firstly, some of the applications of the mapping and localization task in robotics will be presented, talking about the autonomous mobile robots present nowadays and the different methods they use for its localization. After that, it will be explained the problem present currently in the field, which is the Simultaneous Localization And Mapping (SLAM) problem.

1.3.1 Autonomous mobile robots

One of the main application of localization and mapping of areas is the autonomous mobile robots. In the Figure 1.2 we can observe the scheme of how the control of a mobile robot works. It's clear that the part of sensing, information extraction and map building is a very important part of this control, tasks that are tackled under the localization and mapping concept.

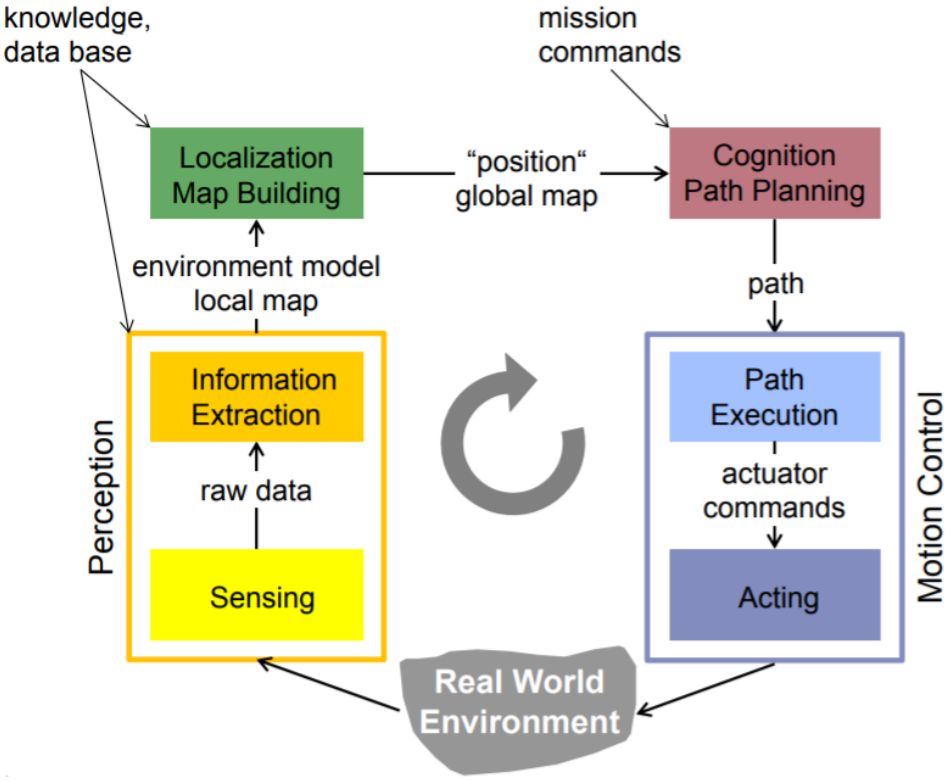

Figure 1.2 *Mobile robot Control Scheme.* ***[3]***

We can find autonomous mobile robots nowadays almost everywhere, from our houses to the space. We find autonomous robots by land, sea and even in the air. Some examples of the autonomous mobile robots where mapping and localization is used will be described in the next pages.

1.3.1.1 Unmanned Aerial Vehicles

Nowadays, one of the most important fields where the mapping and self-localization has experienced a huge grown has been in flying robots with the development of the UAVs (Unmanned Aerial Vehicle) or also known as drones. The usage of them covers a lot of possible applications, like photography, transport and delivery, security, war, farming, sports or simply like an entertainment. The amount of application in the last years have been raising and everyday this technology is applied in a new field. This increase and the commercialization of the technology has caused that, in some countries, already exist regulation related to drones' utilization.

Figure 1.3 *Drone Phantom 4, one of the most advanced commercial drones nowadays.*

The technology underneath the UAVs involves a lot of systems related with communication, engines, mechanics and, of course, mapping and localization of the drone is a crucial task to the correct working of the system and in the interaction with the user in the way that he has to decide where the drone is going in the environment.

Drones nowadays use different methods to implement the localization. The simplest ones usually don't use GPS, they just use visual tracking to get the parameters they need for the localization. However, the more advanced drones incorporate GPS to improve the localization and develop another more complex features.

1.3.1.2 Submarine Robots

Under the seas, there are a lot of tasks that a robot could fulfill better than a human, that's why it's also interesting the implementation of submarine robots. The Autonomous Underwater Vehicles (AUVs) can work under conditions of pressure and available oxygen that the humans couldn't do. Like the drones, they have a lot of applications, mainly military applications, but they could also work like a hobby or for the research in the sea.

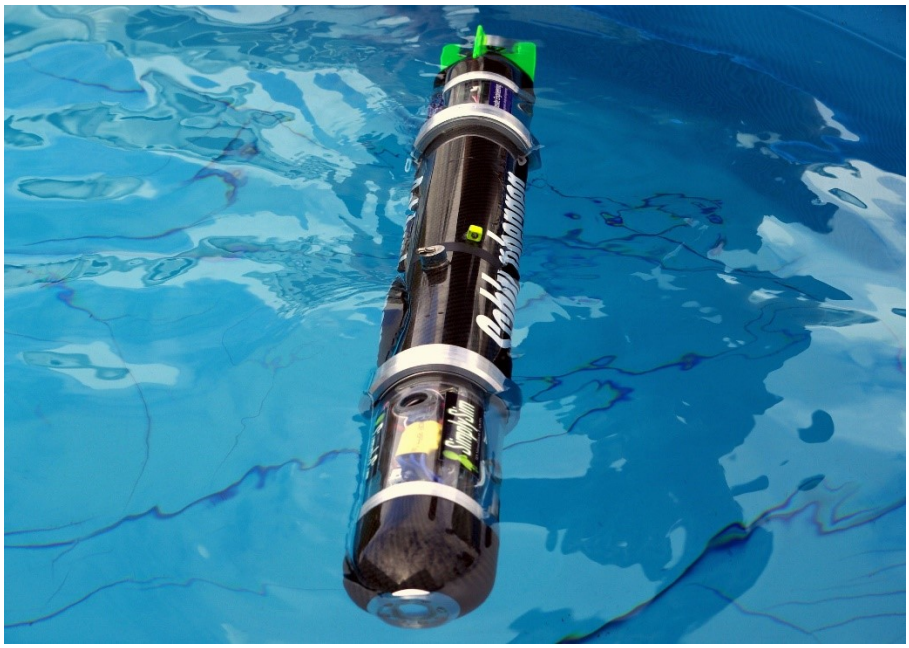

Figure 1.4 *Blackghost AUV designed by Cambridge Autonomous Underwater Vehicle.*

Localization and mapping is also very important for the AUVs, but it's more difficult to tackle than in the drones. This is because the most common sensors used for the localization are not available under water, so another solution has to be developed. For example, the GPS system, that use a lot of robots for their localization, is not available under water because of the impossibility of the transmission of the electromagnetic signals under the water.

1.3.1.3 Land-based robots

This kind of robots is the most common autonomous robot. We can find them in our daily life realizing several tasks with very different forms. There are humanoid robots (as the one used for this thesis) that are more focused to the treatment with humans and other robots more functional that don't have human aspect and are focused on other kind of tasks. Following there will be described some examples of different land-based robots and a brief explanation of its working.

A good approach to the design of robots is trying to imitate some characteristics of the nature. This happens with some robots used for transport of goods. These robots need to be very stable against disruption in the environment, since they may be used for instable environments. One example is the development that the company Boston Dynamics has made in the last years with their dog-based robots and, later on, based in human forms. In the Figure 1.5we can see some of the models.

Figure 1.5 *Robots developed by Boston Dynamics.*

Regarding one of their last models, Spot, it can carry 45 kg with a weight of just 75 kg and its battery last for 45 minutes. It's very important for this kind of robots to maintain the balance to do a good recognition of the environment and, for that task, Spot counts with stereo vision as well as Light Detection and Ranging, which is a method that calculates the distance by measuring the reflected pulses emitted by the target to measure after illuminating it with a laser light. [4]

Between the most common robots in the daily life, we can find robots doing domestic tasks as it could be looking after old people, toys, fixing the garden or cleaning the floor. Maybe, the most notorious and commercialized in the last years have been in the last group, the vacuum robots. We can see one of them in the Figure 1.6.

Figure 1.6 *Vacuum robot (model iRobot Roomba 681).*

In this case, the robot need its localization inside the house to know the distance to his point of charge in order not to run out of battery, and also to know which areas has already cleaned. It achieves that without any camera, just creating a map of where it has been moving and the limits of the environment with a proximity sensor, and saving it the data in the memory. It's another way to locate himself in the environment and the limits of it, less productive, but for this case it's enough for the task that has to be tackled.

Finally, another field where a lot of robots have been developed is in the research area. In this case, the robots are used to investigate and develop features and algorithms that extend the number of applications of robotics.

One example, apart from NAO robot of course, could be TurtleBot. TurtleBot is robot kit development consisting in a mobile platform where it can be added some extensions to develop new programs in autonomous mobile robots. It's a very versatile and low-cost platform and it counts with open-source code, which makes TurtleBot a good option for development of projects with robotics.

Figure 1.7 *TurtleBot 2.*

1.3.2 Simultaneous Localization and Mapping

After defining some examples of mobile autonomous robots and how they face the mapping and location task, we will briefly explain now the problem of the Simultaneous Location and Mapping in robotics or SLAM problem.

SLAM is the problem that refers to simultaneously try to localize the entity that realizes the mapping in the environment while, at the same time, mapping the structure of this environment. The solution of this problem can be implemented with different algorithms and nowadays it has become a very important topic in robotics and other fields of study. [5]

An important feature about the problem of SLAM is that it has to operate in real time, this means that the processing about one camera income must be realized before the next one arrives. In fact, SLAM is mainly a optimization problem of the algorithms used to obtain the best result of the pose.

In the Figure 1.8 we can observe where the SLAM problem is located in the autonomous mobile robotics field. The scan-to-scan matching leads to an accumulated location of the robot estimating it between scans, but forgetting about the existing map, and in the can-to-map matching the scanned data is used to calculate the position of the robot in a map previously stored. [6]

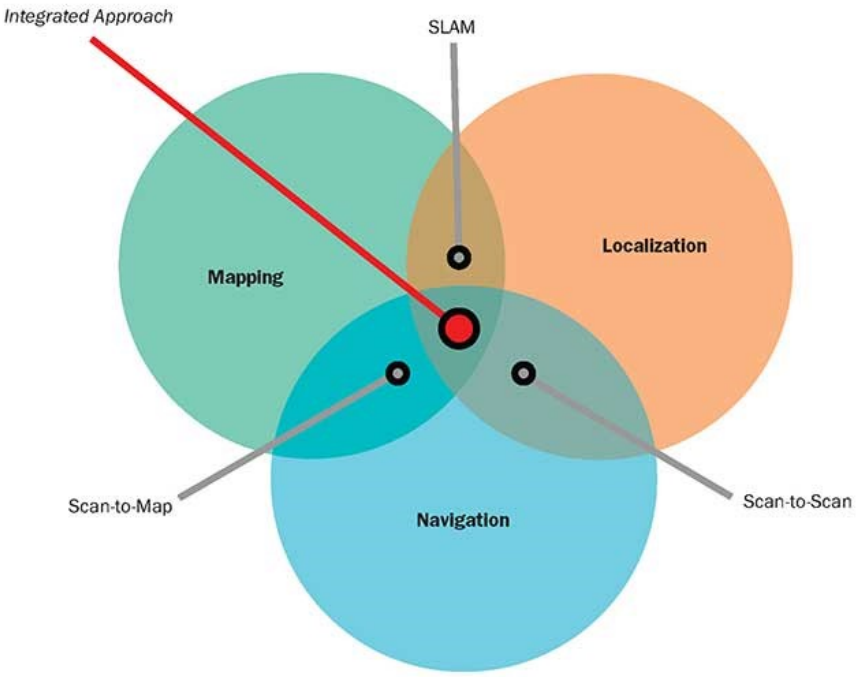

Figure 1.8 *Integrated approach of mobile robots' navigation.* ***[6]***

This thesis will address the localization and mapping task in robotics, which is an essential part for the solution of the SLAM problem.

1.4 Document structure

In this section, the structure of this document is defined in order to make clear the information that will be described in the different parts.

- In the first section, we have done an introduction to the project, explaining the motivation and objectives of it and described the state of the art in the field of interest.
- We will continue in the second chapter with the description of the different tools used for the development of the thesis. The hardware and software of NAO robot will be introduced as well as the NAOqi API and the landmarks used for the localization.
- After that, in the third chapter, it will be described the methodology and implementation of the solution proposed to achieve the objectives of the thesis, explaining also the theoretical foundation of it.
- The fourth chapter will be focused on the experiments carried out to test the reliability of the program developed. The results of different studies and experiments will be described and afterwards they will be analyzed.

- Finally, in the fifth and last chapter, there will be a conclusion to sum up all the content all the document, a discussion whether the objectives were reached or not, and we will finalize with a description about possible future work related with the thesis.

2 TOOLS USED

In this chapter, the different tools used for the development of this thesis will be described in order to have a close idea about them and its role in the project. First of all, we will explain the different parts of hardware and software of the humanoid NAO robot, following by an introduction to the NAOqi API and ending up with the tool we used to set up the environment and the realization of the experiments, the NAO landmarks, also known as Naomarks.

2.1 NAO Robot

The most important tool that has been used for this project is, of course, the NAO Robot. NAO is a humanoid robot developed by the French company Aldebaran Robotics intended for students and researchers. Aldebaran has been updating this model since in the year 2006 the first NAO was developed [7]. Around 10000 NAO robots already sold and used by researchers in 70 countries around the world are some of the achievements of this humanoid robot. [8] The model used for this project has been the NAO fifth version (EVOLUTION) with the body type H25, which is the most complete and up to date NAO model that can be found nowadays. For the development of this project and the experiments two robots of this model have been used (a blue one and an orange one).

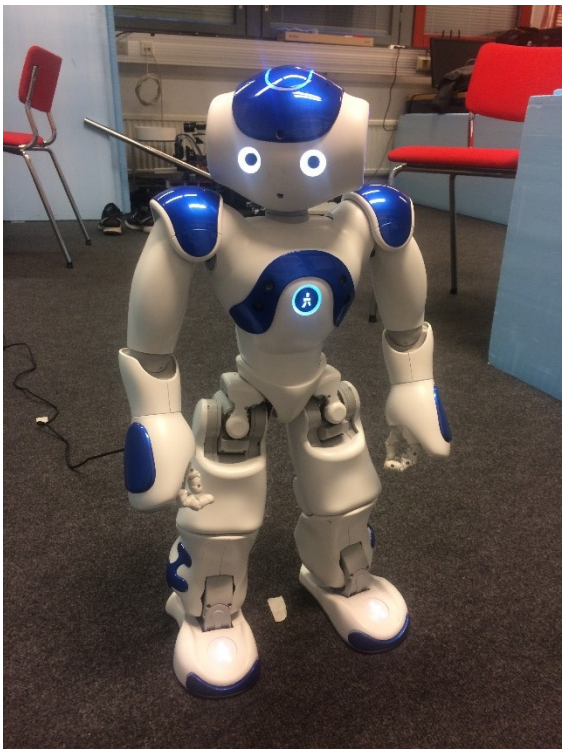

Figure 2.1 *NAO robot model V5 used for the project (blue).*

On the last updated of NAO's robot, NAO EVOLUTION, Aldebaran improved the functionality and operating system of its humanoid robot, whose specifications can be found in the Table 1.

Table 1 *Properties of NAO robot.* ***[9]***

Height	57.4 cm (centimeters)
Weight	5.4 kilograms
Battery	lithium battery with 48.6 Watt-hour
Autonomy	60 min (active use) and 90 minutes (normal use)
Connectivity	Ethernet, Wi-Fi and USB (Universal Serial Bus)
Motherboard	Intel Atom Z530 1.6 Gigahertz, 1 Gigabyte Random Access Memory, 2 GB Flash memory and 8 Gigabyte Micro Secure Digital High Capacity
Built-in OS (Operative System)	NAOqi 2.0
Compatible OS	Linux, Windows, Mac OS
Programming languages	C++, Python, Java, JavaScript
Degrees of freedom	25
Motors	14

Regarding the interaction of the robot with the environment, between all of them, we can find the following elements [9]**:**

- Two video cameras in the forehead.
- Stereo broadcast system formed by 2 speakers in the ears.
- Four microphones in the head.
- Two infra-red sensors located in the eyes.
- Tactile panel in the top part of the head consisting in 3 capacitive sensors and another 3 capacitive sensors in each hand.
- Four force sensitive resistors in each foot and one bumper in each.
- Two ultrasonic sensors in the chest.
- Two switches in the foot working as bumpers and another one in the chest as a button to interact with the robot.

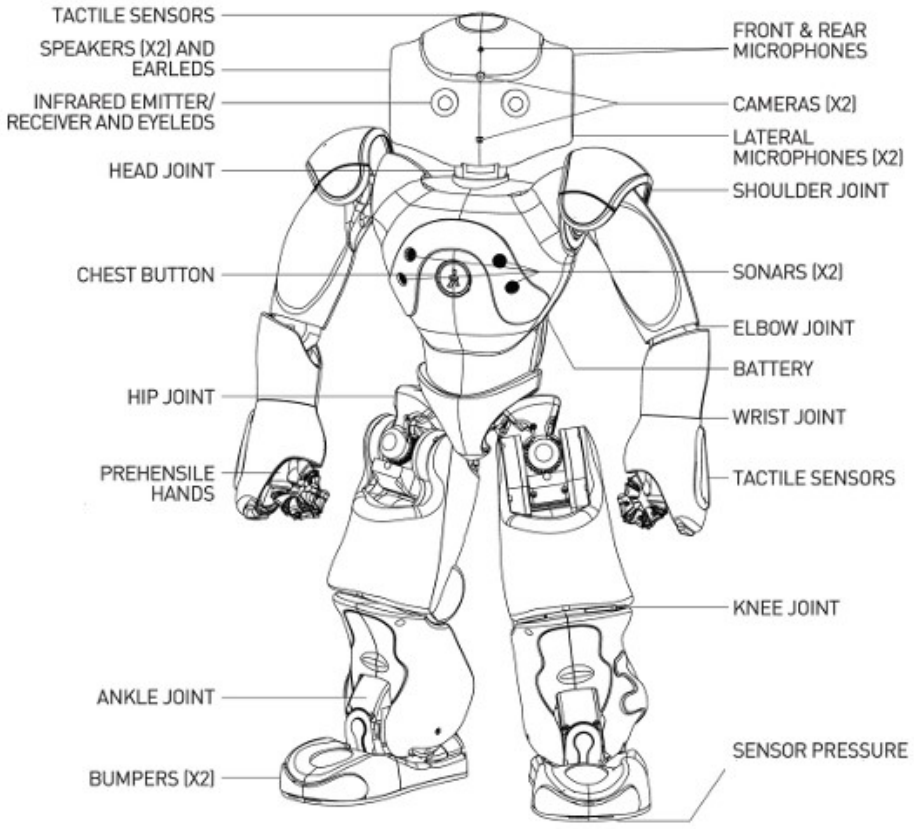

Figure 2.2 *Sensors of NAO Robot body type H25* ***[9]***

Between all the sensors and actuators present in NAO robot, the most important ones for the development of the project will be of course the video cameras and the effector responsible of the head movement, since the robot will do a 180 º (degrees) movement to search for all the landmarks presents around it. In the next pages, there will be more information about them.

In the head of NAO Robot, we can find two identical video cameras, one on the top and one on the bottom of the head. They are both mainly used for detection of objects and landmarks, but in the case of the bottom camera it's also used for NAO's dribbles. The resolution provided is 1280x960 pixels with a frequency of 30 frames per second. In the Figure 2.3 the positioning of both cameras and some geometrical aspects are represented. It's interesting for the project the vertical angle of view of the camera (47.64º) to see the viewing scope of the robot, and the horizontal one (60.97º) for the implementation of the movement of the head to search for landmarks.

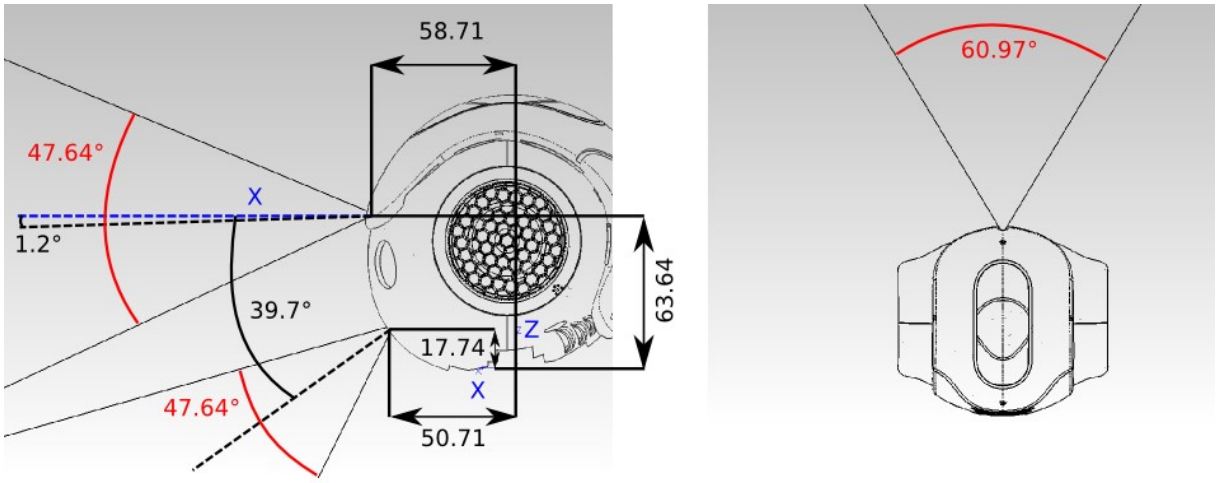

Figure 2.3 *Geometry and angles of view of cameras on NAO head from the side (left) and from the top (right).*

In the Table 2, we can find more technical specifications about the camera, like the resolution, the focus range or the signal/noise ratio. In general, we could say that the specifications of the camera installed in NAO Robot meet the requirements needed for achievement of the goals of this thesis.

Table 2 *Specifications of NAO camera.* ***[9]***

Camera	Model	MT9M114
	Type	System on a Chip Image Sensor
Imaging Array	Resolution	1.22 Megapixels.
	Optical format	1/6 inches
Sensitivity	Pixel size	1.9µm (micrometers) *1.9µm
	Dynamic range	70 dB (decibels)
	Signal/Noise ratio (max)	37dB
	Responsivity	2.24 volts/Lux-sec (second) (550 nanometers)
Output	Data Format	(YUV422 color space)
	Shutter type	Electronic Rolling shutter
View	Focus range	30cm ~ infinity
	Focus type	Fixed focus

Regarding the movement of the head, there are two joints that can be moved and both are used for this project. We can see them in the Figure 2.4, the pitch (frontward and backward movement) and the yaw (horizontal movement of the head).

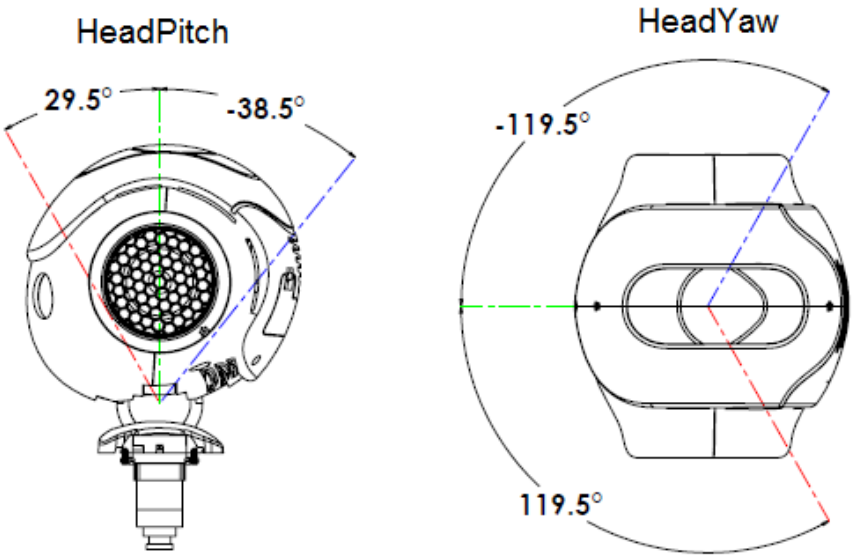

Figure 2.4 *Two joints of the head of NAO Robot: pitch (left) and yaw (right)* ***[9]****.*

The pitch movement will just be used to adjust the head in its central position so the camera is showing us what is just in front of the robot. Furthermore, the yaw joint can carry out movements of 239º, but in the project there will be just used it for do a 180º search of landmarks.

2.2 Choregraphe and Monitor

Another tools used for the development of the project have been Choregraphe and Monitor. Aldebaran Robotics provide this software to help a beginner user to get an easy introduction and understand some of the processes of the robot. Those two programs were very useful in the first weeks working with the robot and for getting a general overview of the possibilities present on working with it.

Choregraphe is an application that allows the user to create programs, using a block-based programming, and test them on a simulated or real robot while monitoring it. It's useful for beginners since there is no need to know to program in any language to develop new applications and try them in the robot, just using the libraries of boxes present in Choregraphe.

The group of actions that can be programmed using Choregraphe cover for instance movement of joints for walking, interaction with the user using the speakers and microphones, or email sending. It's also possible to modify the pre-defined boxes of functions, which are coded in Python. However, if it's needed a deeper and more complex programming we should recur to other means like Software Development Kits.

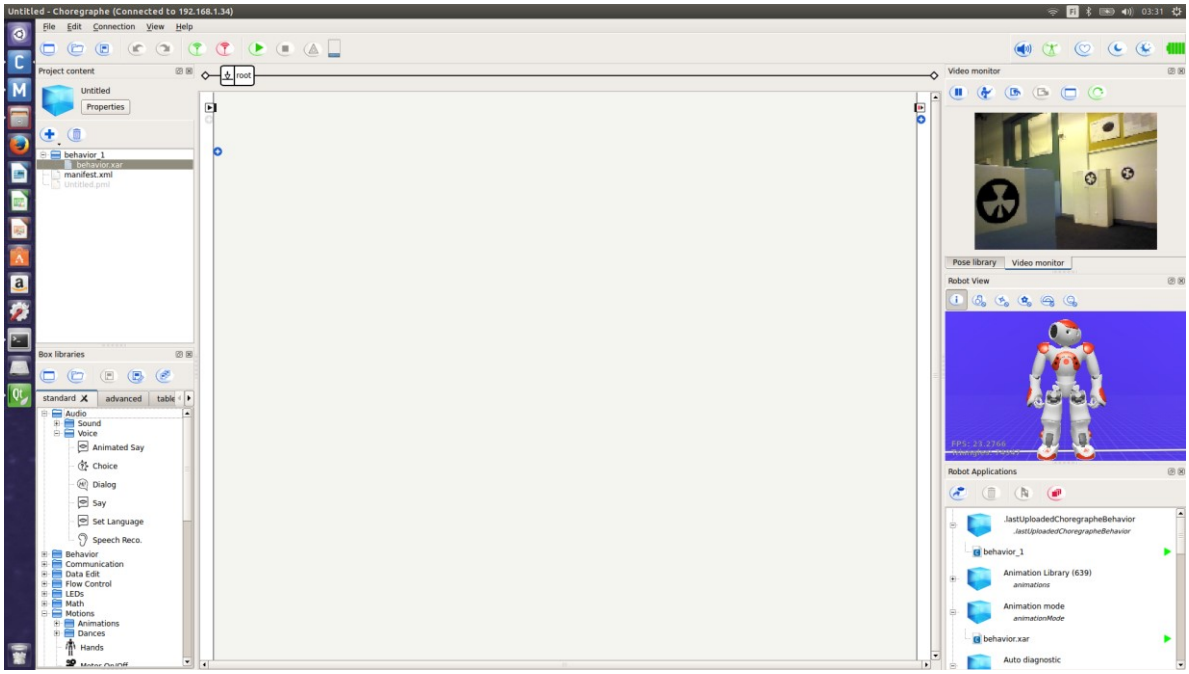

Figure 2.5 *Choregraphe environment.*

In addition to Choregraphe, another tool that has been used during the thesis it's Monitor. Monitor is a software provided to test and configure the camera on NAO robot, which is the main part of NAO robot that has been used during the development of the project. Despite Choregraphe has just been used during the beginning of the project and once the robot and its processes were understood the development of the final program was carried out using the SDK, Monitor has been useful during all the stages of the practical part of the thesis.

The reason why Monitor has been used more is mainly because of its big utility provided by the real-time detection of landmarks, very important to understand the origin of some of the mistakes in the results of the landmark search and how to solve them in the code. Monitor has also another interesting functionalities for beginners, like face recognition or ball detection. Another aspect to take into account of this program is that is possible as well to change some of the parameters of the camera, like the saturation, the brightness or the contrast.

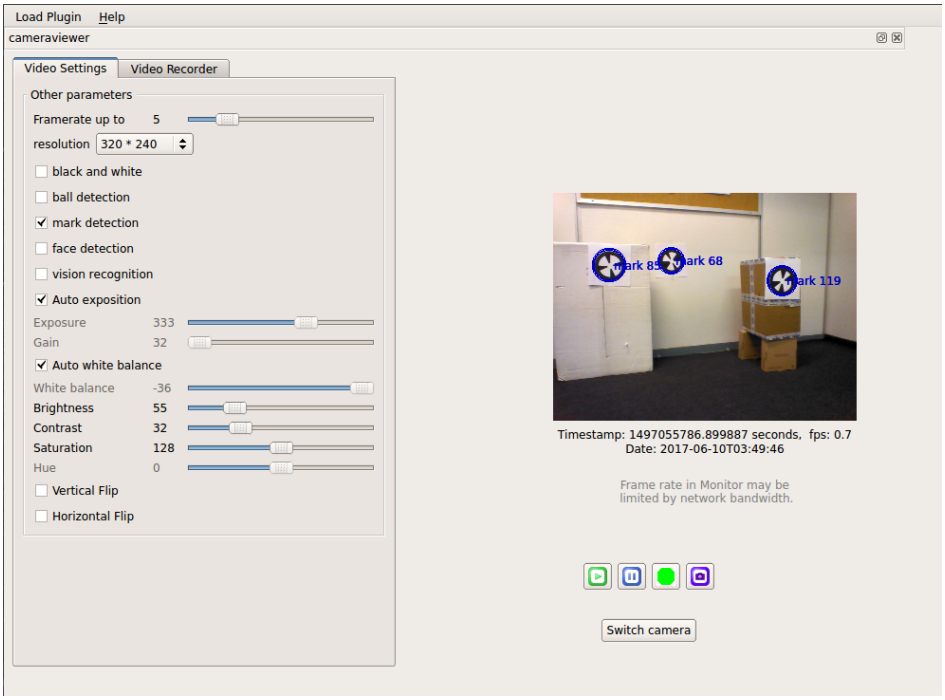

Figure 2.6 *Monitor interface.*

2.3 NAOqi framework

A very important tool used in the project has been the NAOqi framework. NAOqi is the name of the operative system that runs in the robot, and the NAOqi framework is the programming framework provided by Aldebaran Robotics to create applications for NAO robot. The NAOqi framework includes different APIs with functions related to the same topic for the development of projects and it provides solutions to common needs like events management, synchronization, parallelism, etc. The main characteristics of the framework are the following [9]:

- It is a cross platform framework, which means that the development with NAOqi APIs is accessible for several different operative systems, as Linux, Windows or MacOS. The elaboration of this thesis has been accomplished in Linux.
- NAOqi framework is also available for different languages since it's a cross language framework. We can use the NAOqi APIs in C++ and Python, both programming methods are very similar. In this thesis, the programming language will be C++, as we said before.
- It is possible to have available exactly the methods from one executable in another robots as local methods just connecting them using their IP (Internet Protocol) address and port.

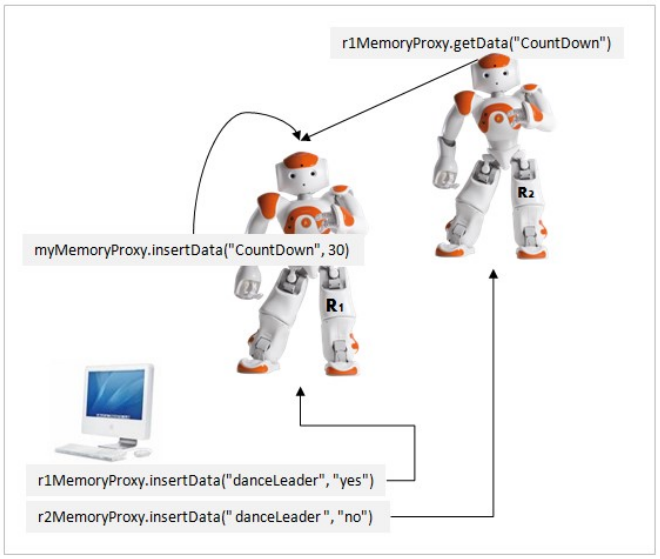

Figure 2.7 *Distributed applications in NAOqi framework.* ***[9]***

The NAOqi framework is structured in different libraries containing modules with functions, methods and classes in each of them. The way to access to these methods and functions is using a broker that load a preferences file that defines the libraries that has to be loaded and, therefore, the modules and methods associated to it.

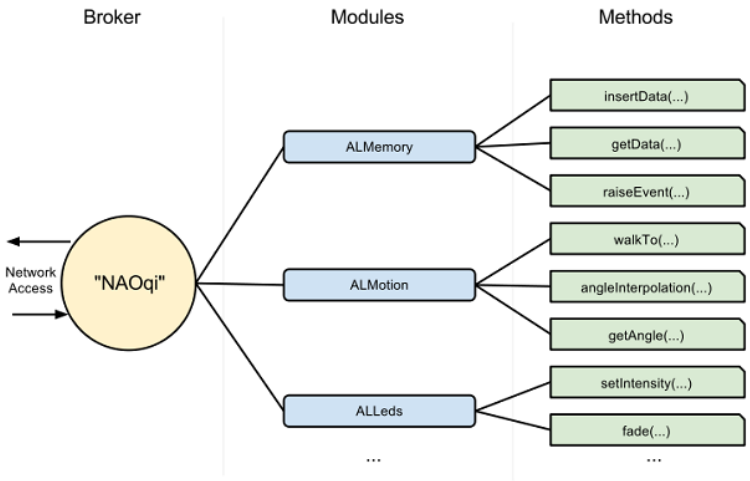

Figure 2.8 *Access to methods using Broker in NAOqi framework.* ***[9]***

These brokers work by their own, so it's not something important to take into account when coding. The most usual element we will be working with while programming the robot will be the proxies, which are objects that behaves as the module it represents, with all the methods belonging to this module.

Another interesting feature of NAOqi framework is that it allows the user to add new modules and new classes and functions, extending this way the API. When creating a new module with new methods, they have been bound in order to be added in the API and to have some benefits like be able to call the function in C++ and Python or execute it locally or remotely. This was very useful to develop some new classes during the project and thus having the possibility to run this methods and functions inside the modules in different robots without the need of code them again.

Between the modules used in the project, we can find the following:

- Some of the modules from the core group like ALMemory or AlValue for basic use of the variables of the program and to access to some date of the memory of the robot.
- From the group of motion, it has been used the module ALMotion, which contain functions for moved the selected joints and configure the speed of movements and other parameters. It has been used for the pitch and yaw movement of the head of NAO.
- Furthermore, most of the functions and methods used in the project come from the vision group, utilizing the module of ALLandmarkDetection, which provides a wide range of useful functions to manage the landmark detection, such as amount and id of landmarks detected, events callbacks, size of the landmark in NAO's camera, etc.
- For a rapid and more agile test and debug of the code in the robot, it was interesting to interact directly with the robot, using for this purpose modules as ALTouch (methods for the head tactile buttons), or ALTextToSpeech.
- Finally, some functions from the math methods have been utilized as well, for work with matrixes and angles operations.

2.4 C++ SDK

Aldebaran also provides in the documentation for NAO robot a Software Development Kit for programming with C++, which allows us to create projects using the NAOqi framework on the PC. The SDK les us develop our program using our preferred IDE (Integrated Development Environment) to edit and debug our code, like Visual Studio, Eclipse or Qt Creator. The last one was the chosen one for this thesis.

2.5 Qt creator

Qt creator is the Integrated Development Environment that has been chosen for this project running in a Linux operating system. It has been elected because of the familiarity with the program and its intuitiveness and clarity for editing the code.

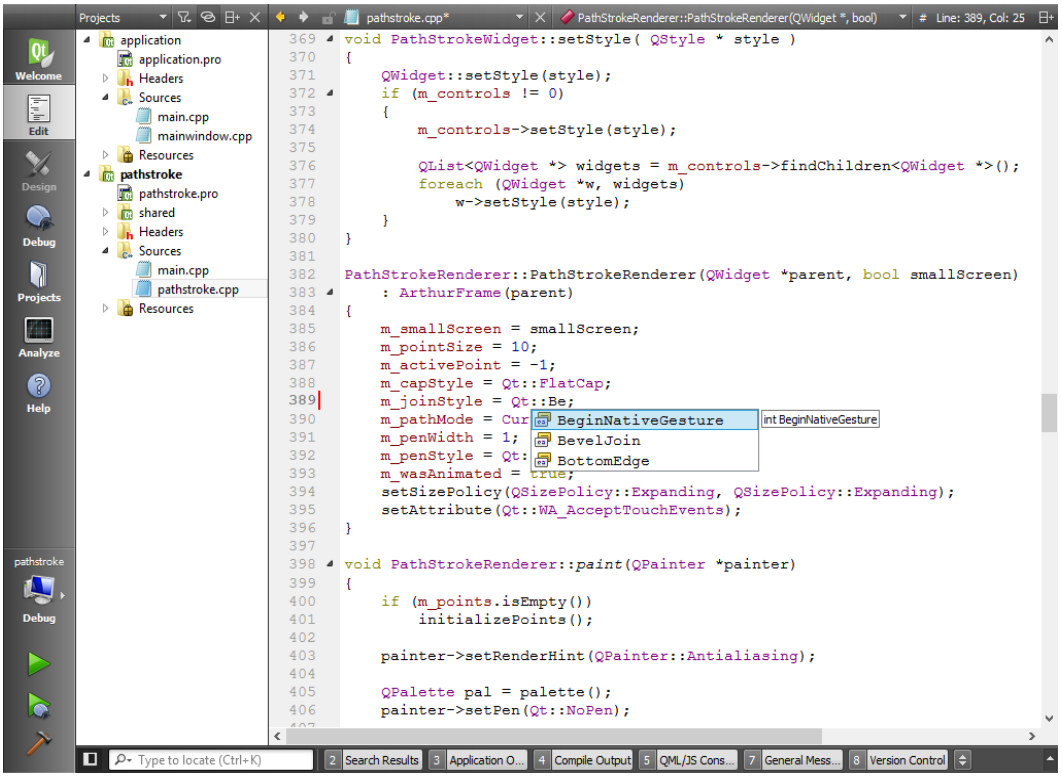

Figure 2.9** Interface of Qt Creator. **[10]

Qt Creator is available for several operative systems and platforms and it offers support for C++ and QML (Qt Meta Language) code edition. Like most of the IDEs in the market, it can compile and run the code of the program, debug it creating breakpoints and monitoring the value of the variables, manage the folders and different files of our project in a clear way or give us hints and advices about the code, between other functionalities.

2.6 Naomarks

Finally, other considerable instrument used for the development of the algorithm and mainly the test realized to it has been the Naomarks or landmarks. This landmarks are black circles with a white shape composed by triangles inside of them that NAO robot can recognize relying on contrast differences. Each of them has its own pattern, depending on how the triangles are situated, and its own identifier, consisting on a number.

The landmarks were used to replace the different objects that NAO could be trying to detect in an environment that has to be mapped. The reasons of this is because otherwise the complexity of the thesis would be too high if we wanted to detect different kind of objects. In that case, there should be needed a training and configuration of the robot for the detection of this project, but with naomarks this is not needed, since the NAOqi API provides that.

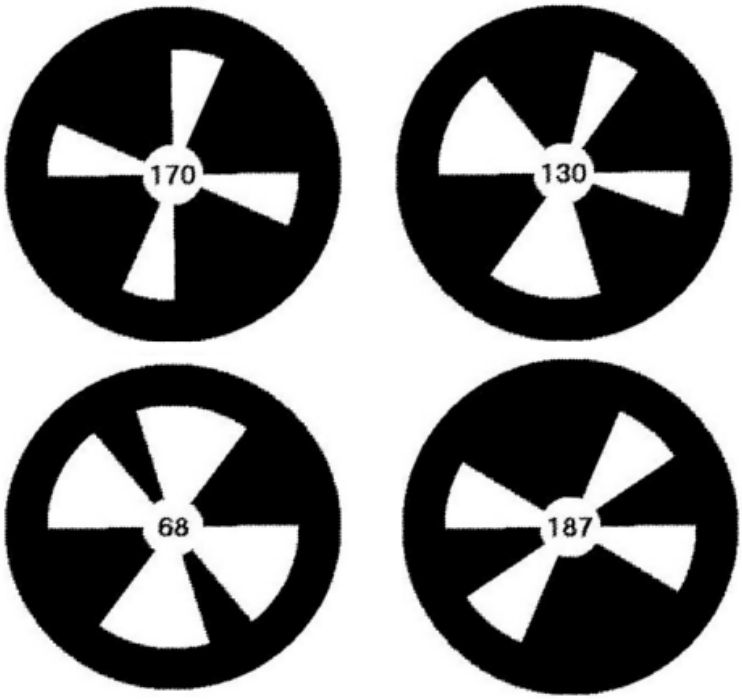

Figure 2.10 *Naomarks.* ***[9]***

Even though the landmark system is well implemented by Aldebaran in NAO robot, we have to consider that it has some limitations: [9]

- Regarding lighting, the landmark system works properly under office light condition, it means between 100 and 500 lux. Since the experiments have been carried out in the laboratory and the project is just intended for indoor environments, the lighting is not a problem for the development of the thesis.
- There is also a limitation in terms of size of the landmark in the image captured by the camera of NAO. The minimum size is stablished in 14 pixels in a QVGA (Quarter Video Graphics Array) image and the maximum in 160 pixels. It means that if the landmark is too close of too far, for the same resolution of the camera, NAO won't be able to recognize it.
- The last limitation applicable is the limitation of the tilt, because if the plane of the camera and the plane of the landmark are not parallel, NAO may have some problems in the recognition of the landmark. The limit on the difference of tilt between these two planes is 60º, over 60º NAO probably won't be able to localize the landmark. However, the rotation of the circle in the image doesn't affect NAO's recognition of the Landmark, it doesn't matter if it's on its original position or outside-down.

Despite all these limitations with the landmarks, we can conclude that they meet the requirements for use them in our project. We just have to take care in the experiments especially that the tilt and size limit are not exceeded.

3 METHODOLOGY AND IMPLEMENTATION

In this chapter, we'll explain the methodology of the process followed to achieve the goal of the project and to programme the final code. We'll talk about the theoretical and mathematical principles behind it, as well as the coding solutions for the problem.

3.1 Distance measurement

The first issue that has to be tackled is how to measure the distance between the robot and each of the landmarks. For that purpose, first we'll define what information does the ALLandMarkDetection module methods can provide us and how is it accessible, afterward we'll describe the method to calculate the distance using these parameters of the landmark and finally we'll describe the movement of NAO's head to search for all the landmarks visible in its scope and how the event of a new landmark it's treated by the code.

3.1.1 Information about the landmark detected

The NAOqi framework provides us, in the ALLandMarkDetection module, with methods to obtain the information of the landmarks that NAO robot is seeing. The results are collected in a variable in ALMemory, accessible with a proxy to this module, and it's an array of elements (empty when no landmark has been detected). The data that is saved in this variable and its structure is the following. [9]

$$[\,\boldsymbol{TimeStampField}\,]\,[\,\boldsymbol{Mark\_info\_0}\,,\boldsymbol{Mark\_info\_1},\ldots,\boldsymbol{Mark\_info\_N-1}\quad \textit{(1)}$$

In equation 1, we must consider that:

- TimeStampField is the time stamp of the image used for realize the detection. We won't use this variable for the project.
- N is the total number of landmarks that NAO has detected with the camera. We have the information of all of them in the array.

Mark_info of each landmark detected is organized in the following way:

$$\boldsymbol{Mark\_info} = [\,\boldsymbol{ShapeInfo},[\boldsymbol{MarkID}]\,] \qquad \textit{(2)}$$

As we can see, we find another array inside the Mark_info variable, where the second element is the id of the landmark detected (the number of it) and the first element is the information about the shape of the Naomark. This element contains the data we'll need to calculate the distance to this landmark, and its structured as follows:

$$\boldsymbol{ShapeInfo} = [\,\boldsymbol{1, alpha, beta, sizeX, sizeY, heading}] \quad \textit{(3)}$$

Therefore, we'll get from the ShapeInfo array information needed to calculate the distance to the landmark, consisting in:

- On the second and third element, there are alpha and beta, whose value is the vertical and horizontal location respectively of the center of the landmark with respect to the center of the image expressed in terms of the camera angle, expressed in radians [11].
- The sizeX and sizeY, if we look at the third and fourth component of the array, correspond to the size in the X and Y direction respectively of the mark, expressed in radians as well.
- Finally, the variable of heading describes the orientation of the landmark with respect to the vertical axis. This value will not be used in the project.

3.1.2 Distance to landmark

Once we have collected all the data needed from the landmark to obtain the distance between the robot and the mark, the next step is to explain the calculation of this distance. For this purpose, the values needed will be the alpha, beta and the sizeX, obtained in the last section.

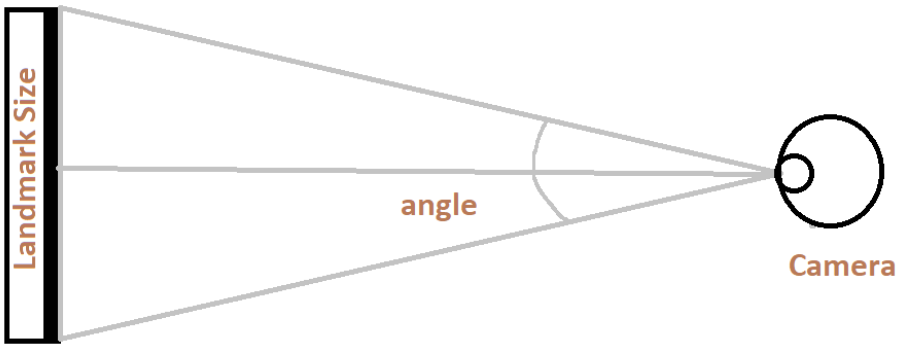

Figure 3.1 *Triangle formed in the detection of the landmark.*

If we look at the Figure 3.1, we can deduce the following expression:

$$\boldsymbol{tan(angle/2)} = \frac{\boldsymbol{LS/2}}{\boldsymbol{D}} \quad \textit{(4)}$$

Where *angle* is the vertical size of the landmark in radians of the NAO camera (sizeX variable, obtained from the ALLandMarkDetection module), *LS* is the real size of the printed landmark and *D* is the distance to the landmark. Then, we can find the value of the distance from the Equation 4, ant it's:

$$D = \frac{LS/2}{tan\left(\frac{sizeX}{2}\right)} \quad (5)$$

With the Equation 5 we would obtain the distance to the landmark, but in the camera frame. So, to obtain the distance to the landmark in the robot frame, we need to change from the camera frame to the robot frame. For that, we need to get the transform matrix from the camera to the robot frame, using the position of the camera in relation to the robot frame (which is located between the two legs of NAO in the floor), also the rotational transformation between the landmark and the camera, utilizing the values of alpha and beta, gathered from ALLandMarkDetection module, and finally the translation transform matrix from the landmark to the camera, using the distance calculated before. In the Figure 3.2 we can see how the robot frame in NAO is located and orientated, with the X axis looking towards and the Y to the left.

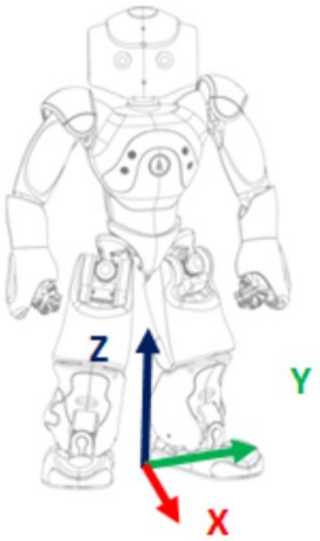

Figure 3.2 *Robot frame of NAO.*

For creating the transformation matrixes, it will be used the built-in functions for transformation matrixes that we can find in the ALmath module, also very useful for operate with them. Therefore, the landmark coordinates in the robot frame will be calculated as:

$$\boldsymbol{LandmarkCoordinates} = \boldsymbol{cameratoRobot} * \boldsymbol{landmarkCameraRotation} * \boldsymbol{landmarkCameraTranslation} \quad \textit{(6)} \; \textit{[11]}$$

In the result obtained with the Equation 6 we would get the position of the landmark in X, Y and Z axis respect to the robot frame, but for this project we will just care about the X and Y relative position of the landmark, since we will just work in 2 dimensions.

3.1.3 Movement of NAO's head

For achieving a general idea of the scenario, it has been developed an algorithm where NAO will move the head in 180º around itself so it can scan all the landmarks present in the environment. In this section, first there will be explained how the movement of the head has been coded and, afterwards, how to tackle the problem of several landmarks appearing in NAO's camera while moving.

In order to move NAO's head, a few methods from the module ALMotion have been used, mainly the methods of setStiffenesses and angleInterpolation. The stiffness of the join has to be 1 if we want that the joint to move as we want with our code order, and after complete the movements we should reset it again to 0 so the joint can be freely moved afterwards.

Once the stiffness is rigid, we'll use angleInterpolation to move the head. This function allows us to set the angle where we want the joint to be situated and to choose the velocity of the movement. We will use two velocities, one fast for the movement to the origin of the scan and one slower to realize the 180º scan, because if we do it very fast it may lead the robot to commit in some failures in the detection. The extremes of the scan are 90º and -90º in relation to the Figure 2.3, which correspond to 1.57 rad and -1.57 rad in the code.

In the Pseudocode 1, we can see the algorithm developed for the movement to the initial position firstly and afterwards for the scan.

```
1  Begin
2  Move head pitch to 0                    //NAO's head totally horizontal
3  If (yaw position is closer to right extreme)
4        Move to right extreme with rapid velocity
5  Else
6        Move to left extreme with rapid velocity   //Configuration completed
7  EndIf
8  If (yaw position is on the right extreme)
9        Move to the left with normal velocity
10 Else
11       Move to the right                  //Scan completed
12 EndIf
13 End
```

Pseudocode 1 *Head movement algorithm.*

It is also important to tackle the detection of landmark while the head is moving so it doesn't just move the head, but also detecting and saving the values of the landmark that sees when moving the head.

For this purpose, NAOqi API offers us a way to create callback functions to react to some events that occur in the robot. Firstly, we have to subscribe to the event called "landmarkdetected" when we are starting the scan of landmarks. It's important not to do it before the configuration of the robot is completed (with the head situated in one of the extremes), because the robot should only look for landmarks when the configuration is done. Also, when the scan looking for landmarks is finished, the subscription to this event has to be implemented. Otherwise, in the next time executing the program, the NAO robot will start to look for landmarks before the configuration is terminated because it will be still subscribed to the event.

Furthermore, we also have to create a function that will be executed every time a landmark appears in NAO's camera. This function must be able to recognize when a landmark has been already added to the landmarks detected list, so each different landmark just appears one time. This is important because the function will be executed several times for the same landmark detected since the time that takes the landmark to disappear of NAO's camera because of the movement of the head it's much bigger than the time of execution of the callback method. Also, it has to be verified that the data of the landmark detected is not empty. This problem is caused when the landmark is in the limit of NAO's image, it could call the function but when we want to access to NAO's memory in order to get the data, the landmark is no longer available and it's empty.

The Pseudocode 2 shows how the callback function for the event "landmarkdetected" has been implemented.

```
1  Begin
2  If (landmark detected ID = 0)
3        exit callback function              //Landmark data is empty
4  EndIf
5  If (landmark detected ID is in landmarks saved ID list)
6        exit callback function              //Landmark already saved before
7  EndIf
8  Save data and update list of landmarks detected
9  End
```

Pseudocode 2 *Callback function for landmark detected event.*

The data that needs to be saved for every landmark for calculating afterwards the distance to it is:

- The marker ID.
- Alpha, beta and SizeX from the ShapeInfo array explained in the section 3.1.1.
- The relative position of the camera in the moment of the detection of the landmark respect the robot frame. This transform can be obtained by using the method of getTransform from the motion module on the NAOqi API.

After this procedure, and the calculation of the distance to the landmark that was explained in the section 3.1.2, if this code is executed in both NAOS, we would obtain the ID of every landmark around each NAO robot in 180º and its distance to itself in the robot frame.

3.1.4 Mistake's correction in landmark detection

When the code defined before was tested in the robots for getting the detection of the landmarks, some errors were found in some of the results. Using the software Monitor and its tool for the landmark detection in real time with the robot, it appeared mistakes in the moment of the detection of the landmark ID. In some cases, the robot confused a real landmark with another one. This mistake may be caused because of the movement of the head and the instability of the image or because of a bad lighting or the failure to fulfill other of the limitations of Naomarks explained in the section 2.6.

Another problem found in the code was that the accuracy of the distance between NAO robot and the landmarks wasn't good enough to ensure a good final result on the mapping task. This could be caused because of the movement of the head while obtaining the transform of the relative position of the camera in the robot frame or because of problem with the accuracy in the methods of the landmark detection module.

To solve these two problems, the solution proposed was to do 3 turns of 180º each to improve the results of the landmark detection. With this solution, the first problem would be solved since the landmarks detected in just one or two turns, which are the incorrect ones, will be deleted, just the correct landmarks that were detected in all the turns will be considered. Furthermore, the accuracy of the landmark distance would be improved, since the distance in each of the turns will be used to calculate the average between the three times that the landmark was detected.

It has been decided that three turns are enough for improving the results exactitude without taking much time to run the program. Furthermore, the use of dynamic memory has not been implemented in this thesis because the number of possible landmarks detected by NAO can't be bigger than 10, so the space of memory used for the landmarks' data storage doesn't affect significantly to NAO's memory. However, the dynamic memory could be implemented in the code as a possible future work for improving the efficiency of the program.

Therefore, in the Pseudocode 3 it can be observed how the callback function for the event "landmarkdetected" has been modified to apply the solution purpose and have a good storage of the data for each landmark and each turn.

```
1  Begin
2  If (landmark detected ID = 0)
3        exit callback function              //Landmark data is empty
4  EndIf
5  If (landmark detected ID is in landmarks saved ID list for this turn)
6        exit callback function       //Landmark already saved before in this turn
7  EndIf
8  Save data for this turn and update list of landmarks detected in this turn
9  End
```

Pseudocode 3 *Final version of the callback function for landmark detected event with the implementation of the solution for exactitude's improvement.*

3.2 Localization of NAOs and the objects in the map

Once we have all the information of the distance between the landmarks and the NAO robots and the data of the location of some known landmarks, the first step in the mapping of the area is to locate the robots in the map and, afterwards, to calculate the location of the unknown landmarks to complete the map.

3.2.1 Localization of NAO Robots

It is important to consider that for the localization of NAO robots in the map we need to know the position in the map (in X and Y axis) of at least two landmarks. The reason why it has to be at least two will be explained later on.

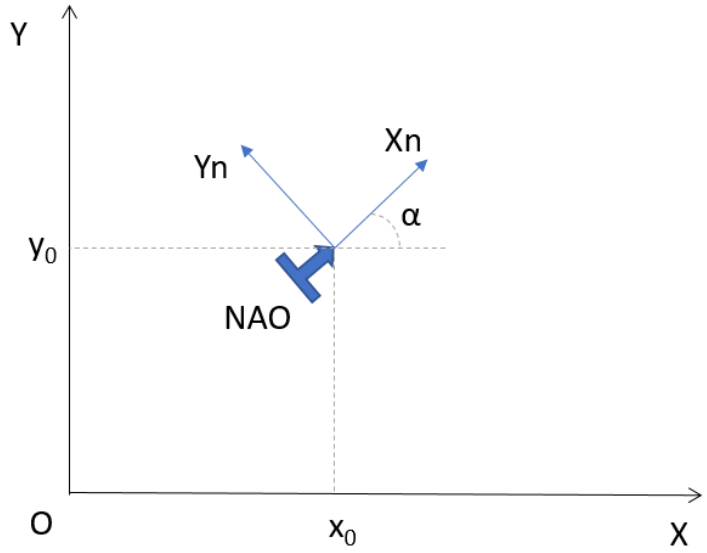

Figure 3.3 *Representation of NAO robot's reference system.*

In the Figure 3.3 we can observe the 2D model of NAO's situation in the global frame. The data that we need to calculate in this section is not only the NAO's x and y positions in the general reference system (x_0 and y_0 in the diagram), but also the angle of the orientation of the robot's frame in relation with the general reference system (α). This is needed to be able later to calculate the position of the unknown landmarks, starting from the relative distance between them and the robot.

For calculating NAO's position in the global frame (x_0, y_0 and α) we'll use some landmarks whose position in the global frame we already know, and the relative position of this landmarks respect to the robot as well. We'll use the transformation matrix for the coordinates between two reference systems in a two-dimensional plane. This transformation matrix can be observed in the Equation 7.

$$\begin{pmatrix} x_{global} \\ y_{global} \\ 1 \end{pmatrix} = \begin{pmatrix} cos(\alpha) & -sin(\alpha) & x_0 \\ sin(\alpha) & cos(\alpha) & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \\ 1 \end{pmatrix} \quad (7) \; [12]$$

In the Equation 7, x_{global} and y_{global} is the position of the landmark in the global frame, x_0 and y_0 represents the position of the robot NAO in the global frame, the angle α is the orientation of the NAO's local framework with respect to the global framework, and x_n and y_n are the coordinates of the landmark in NAO's reference system, obtained with the scan of landmarks. In the Figure 3.4 there is a graphical representation of it. The so-called transformation matrix is the matrix in the middle and we can pass from local coordinates (NAO's framework) to global coordinates just multiplying.

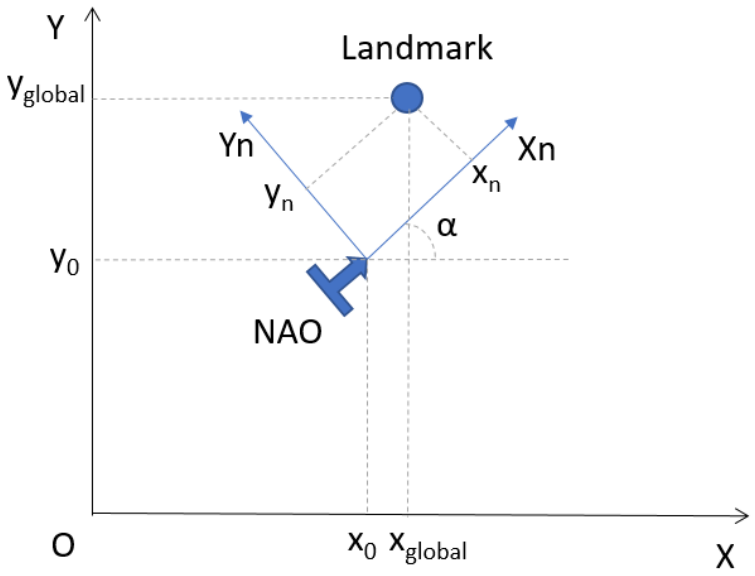

Figure 3.4 *Representation of Landmark in NAO and global framework.*

As we can see in the Equation 7, we know the value of all the variables but three unknown parameters that we need to solve but just with two equations since in the last row (the one corresponding to the third dimension that we don't take into account) the equation is type 1=1. That's the reason why we need at least another landmark whose global location is known in order to solve the location of the robot. Then we would have the following equations:

$$\begin{pmatrix} x_{1global} \\ y_{1global} \\ 1 \end{pmatrix} = \begin{pmatrix} cos(\alpha) & -sin(\alpha) & x_0 \\ sin(\alpha) & cos(\alpha) & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{1n} \\ y_{1n} \\ 1 \end{pmatrix} \quad (8)$$

$$\begin{pmatrix} x_{2global} \\ y_{2global} \\ 1 \end{pmatrix} = \begin{pmatrix} cos(\alpha) & -sin(\alpha) & x_0 \\ sin(\alpha) & cos(\alpha) & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{2n} \\ y_{2n} \\ 1 \end{pmatrix} \quad (9)$$

It is important that these two landmarks' position are not too close because otherwise there will be a big error in the result of NAO localization. This will be caused because we would work with a repeated similar information and we would be close to have the same equation two times. In the chapter 4 this will be analyzed with experimental data.

If we try to clear up these equations for the variables that we're trying to find and we replace sin(α) and cos(α)for a and b respectively, we'll end up a four-equation system, that can also be written in a matrix system like the following:

$$\begin{pmatrix} x_{1global} \\ y_{1global} \\ x_{2global} \\ y_{2global} \end{pmatrix} = \begin{pmatrix} -y_{1n} & x_{1n} & 1 & 0 \\ x_{1n} & y_{1n} & 0 & 1 \\ -y_{2n} & x_{2n} & 1 & 0 \\ x_{2n} & y_{2n} & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ x_0 \\ y_0 \end{pmatrix} \quad (10)$$

For solving the Equation10, we'll use the library of Eigen, which is an open and versatile library for working with linear algebra [13]. Between all the methods that this library offers to solve systems of equations of type Ax=b, it will be used the method of column pivoting. The reason of this election is that this method doesn't require the A matrix to be invertible, so it works for all the possible situations, and it's quite fast.

Once we solve the equation system, we will easily have the position of the robot (x_0 and y_0). For getting the orientation, we will get two variables, a and b. To obtain the angle correctly, first we should make sure that the sin and the cos are between -1 and 1, because in some cases, due to inaccuracy problems in the landmark detection when the angle is close to 0 or π, we could obtain value out of the range. For these cases, we'll just adjust the value of a and b to the closer limit in each case.

Another thing to take into account is which of both variables should we use to determine the angle. Some studies show that the cosine provides a better approximation of the angle [11], so the value of b will be used in the calculation. Finally, we should also tackle the issue of the different quarters, take into account that in the third and fourth quadrant (when the sine is negative), if we are using the cosine to the calculation, the real orientation would be 360º - arccosine(b). In the Pseudocode 4 we can see how the calculation of the orientation for each robot has been finally implemented in the code.

```
10  Begin
11  If (a>1)              //First we check if sin or cos are out of range
12          a=1
13  EndIf
14  If (b>1)
15          b=1
16  EndIf
17  If (a<-1)
18          a=-1
19  EndIf
20  If (b<-1)
21          b=-1
22  EndIf
23  If (a>0)              //If we are in the first or second quadrant (sin>0)
24          Orientation = arccosine(b)
25  Else                  //If it's the third or fourth quadrant (sin<0)
26          Orientation = 360º - arccosine(b)
27  EndIf
28  End
```

Pseudocode 4 *Calculation of the orientation of the robot based on sin(α) and cos(α).*

A few more characteristics of the code regarding the NAOs situation in the map should be mentioned. When we decide to set the location of more than 2 landmarks in the global frame, that is, to have more than 2 known landmarks, the robot will take every pair of landmarks and calculate its position and orientation in the map with them, utilizing all possible combinations of landmarks and, afterwards, calculate its pose with the mean between all the calculated with them. This will improve the localization accuracy of the robot, as we will see later in the experiments.

Furthermore, the code is able to let each of the robots use for its own localization the landmarks that it's able to see. That means that if, for example, one robot just see two of the pre-stablished known landmarks and the other one see three of them, they will both use those one that they see for their own localization in the map, even sharing some known landmarks between them.

To sum up, with this process described, both NAO robots will be able to locate themselves in the map with the help of the known landmarks and they would be able to know where is the other robot in relation with itself and the distance that separates them without the need to see it. In addition, they both will be able to locate all the landmarks around also without seeing them, just with the help of the other robot. This will be tackled in the next section.

3.2.2 Mapping of unknown landmarks

Once we have located the robot with its coordinates in the global frame and its orientation, the mapping of the unknown landmarks it's very easy. We just have to use the transformation matrix from the local robot frame, where we know the distance in both axis to the landmark from the robot, to the global reference system to obtain the location of each landmark in the map. We can see this operation in the Equation 11.

$$\begin{pmatrix} \boldsymbol{x_{global}} \\ \boldsymbol{y_{global}} \\ \mathbf{1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{cos(\alpha)} & \boldsymbol{-sin(\alpha)} & \boldsymbol{x_0} \\ \boldsymbol{sin(\alpha)} & \boldsymbol{cos(\alpha)} & \boldsymbol{y_0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix} \begin{pmatrix} \boldsymbol{x_n} \\ \boldsymbol{y_n} \\ \mathbf{1} \end{pmatrix} \quad \textit{(11)}$$

Where the x_{global} and y_{global} are the variables that we want to solve, α, x_o and y_0 represent the pose of the robot that detected this landmark and x_n and y_n are the relative distance from the robot to the landmark, that was calculated based on the data from the NAO camera.

Now, to get the value of the position of each landmark detected from NAO that is unknown in the map, we just have to multiply matrixes, and we will obtain the following equations:

$$\boldsymbol{x_{global} = cos(\alpha) * x_n - sin(\alpha) * y_n + x_0} \quad \textit{(12)}$$

$$\boldsymbol{y_{global} = sin(\alpha) * x_n + cos(\alpha) * y_n + x_0} \quad \textit{(13)}$$

With the Equations 12 and 13, the algorithm just has to look for the landmarks whose location is not predefined and apply them to get the global location of each landmark of the different robots.

Finally, we have all the elements in the map, all the robots and all the landmarks whose position in the beginning wasn't stablished. This has been possible by the communication and information shared by the robots about their location and the location of the landmarks.

A function has been implemented to export all this data structured in rows and columns to a .csv (comma separated values) file so it's easier afterwards, in the next chapter, to analyze the results of the experiments carried out.

4 EXPERIMENTS AND RESULTS ANALYSIS

The aim of this chapter is to test the program developed in the previous chapters and to test the accuracy of the results obtained. For this aim, it has been carried out several experiments changing different parameters to see the effect to the results in an indoor prepared environment.

4.1 Head movement speed study

First of all, before realizing the experiments in the environment, a decision had to be taken, which is the velocity of the movement of the head of the robots. For that purpose, some tests were carried out to determine which velocity for the yaw movement during the scan of landmarks was the most convenient. In the Table 3 we can see the results changing the time that takes NAO to go over the 180º movement trying to detect 5 naomarks around it.

Table 3 *Results of head speed study.*

Time (sec)	Landmarks Detected
5	3
7.5	4
10	5
12.5	5
15	5

As we can see in the Table 3, with 5 and 7.5 seconds to go over the turn the velocity is too high and NAO is not able to detect all the landmarks present around it. Also, above 10 seconds, with the other two experiments realized, the number of landmarks detected are the total of them. Therefore, the head speed chosen for the scan for all the experiments will be 10 seconds per turn, which is the faster that allow NAO to detect correctly all the landmarks around it.

It is important to remember than in the configuration of the robot to go to its initial position for the scan the speed used will be faster, since there is no landmark detection in this movement. it has been chosen a velocity of 1.5 seconds for the yaw movement and 0.5 seconds for the pitch configuration, so the configuration of the initial position of the robots, in case they are not in this position already, are as fast as possible.

4.2 Environment set up

The experiments have been carried out in indoors, particularly in a laboratory. It has been created an area where the experiments are carried out that meet the requirements of our thesis, which means that both of the robots have a barrier so they cannot see each other and that some landmarks are visible just for some of them. Furthermore, in this area they could both detect some common landmarks. For the experiments it has just been used two NAO robots. In the Figure 4.1 we can observe the characteristics of the environment.

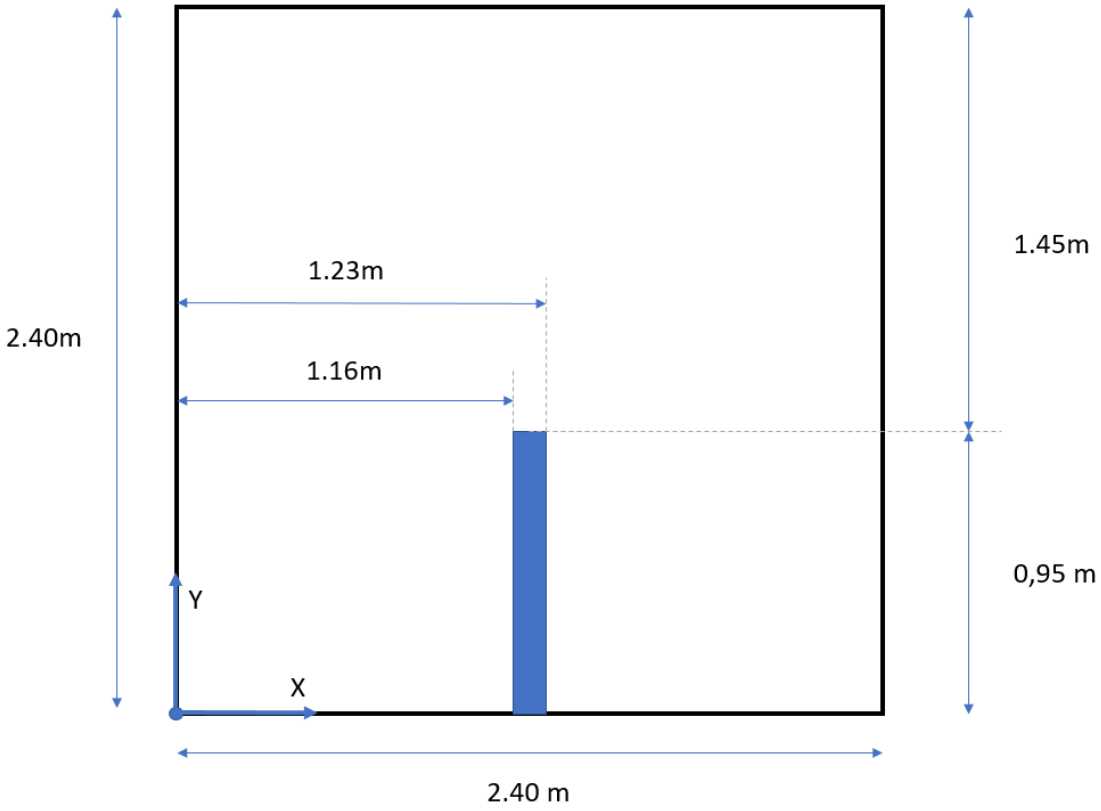

Figure 4.1 *Scheme of the experiments' environment dimensions.*

As we can see in the Figure 4.1, the experiments area is a 2.40 m (meters) square where it has been installed a wall in the middle of it. In this construction, the robots should be located in the bottom part of the image, so the wall is between them and they cannot see each other. There is a common area in the top part of the construction where some common landmarks that both robots can detect could be located.

The origin of the global coordinate system has been arbitrary located in the bottom part in the left. This way, all the coordinated present in the map will be with positive value in the global frame, so it makes the work easier.

In the Figure 4.2 there are two pictures that show how this map has been implemented in the laboratory with some of the landmarks.

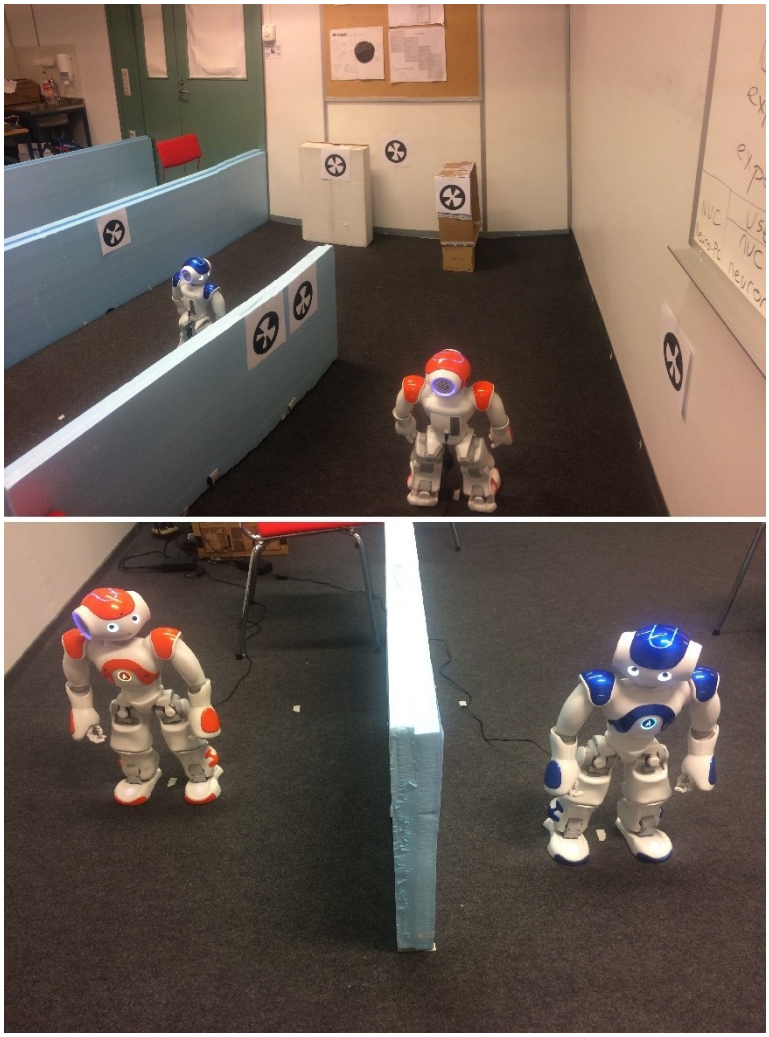

Figure 4.2 *Implementation of the experiments' environment on the laboratory.*

As we can see in the Figure 4.2, the landmarks have been positioned in all the map with stickers that can be easily removed to change the conditions of the environment and see how this affect to the results. It's important to define the size of the landmark, it is, the diameter of the black circle. For the experiments, it has been established a size of 15.5 cm, which has been added to the code, but it could be easily changed depending of the needs of future experiments.

4.3 Experiments realized and results

4 experiments were realized to see the influence in the accuracy of the results of some drivers like:

- Number of common known landmarks for localization of the robots.
- Distance between the landmarks used for the localization of both robots.
- Distance from the robots to the landmarks used for their own localization.

In all the experiments both robots are located in both sides of a wall so any of them can see each other and the known naomarks will be visible for both robots to simplify the experiments.

4.3.1 First experiment

In this first experiment, it has been used three know landmarks with a good distance between them for robots' localization in the map. In the Table 4 we can see the real position of the robots and the landmarks in the map and the calculated position of all of them.

Table 4 *Results of the first experiment.*

		Real			Calculated		
Type	**ID**	**X(m)**	**Y(m)**	**Orientation (rad)**	**X(m)**	**Y(m)**	**Orientation (rad)**
Robot	bluenao	0.75	0.60	1.57	0.58	0.73	1.50
Robot	orangenao	1.80	0.40	1.57	1.85	0.50	1.65
UnknownLandmark	[84]	1.16	0.84		1.08	0.92	
UnknownLandmark	[107]	0.00	0.50		-0.17	0.94	
UnknownLandmark	[108]	1.23	0.40		1.27	0.51	
UnknownLandmark	[112]	1.23	0.65		1.24	0.78	
UnknownLandmark	[114]	2.40	0.30		2.49	0.46	
KnownLandmark	[68]	1.11	2.40				
KnownLandmark	[119]	1.60	1.89				
KnownLandmark	[85]	0.75	2.15				

If we represent the results in the environment that was built, the results of the experiment can be observed in the Figure 4.3. We can see that the results with three known landmarks separated between them are quite acceptable, obtaining a good approximation of the position of the robots and the position of the unknown landmarks. The value of the averaged absolute error in all the locations of both robots and landmarks is of 12.7 cm, which means a relative error of 5.3 % (percent) in a map of 240 cm.

We can also observe that, when the error is bigger on the localization of the robot, this error is also present in the unknown landmarks that this robot will locate, as we can see in the case of the localization of bluenao, whose localization has a bigger error than the localization of orangenao, and the bad calculation of the landmark on its left (landmark [107]) in comparison with the localization of the naomarks detected by orangenao.

Regarding the calculation of the orientation of the robots, it has been calculated with a good accuracy, the averaged absolute error is of 0.08 rad (radians), which means a relative error of 2.5%. This orientation accuracy is, as well as the robot positioning accuracy (absolute averaged error of 11.3 cm, 4.7%), important for the mapping of the unknown landmarks of each robot.

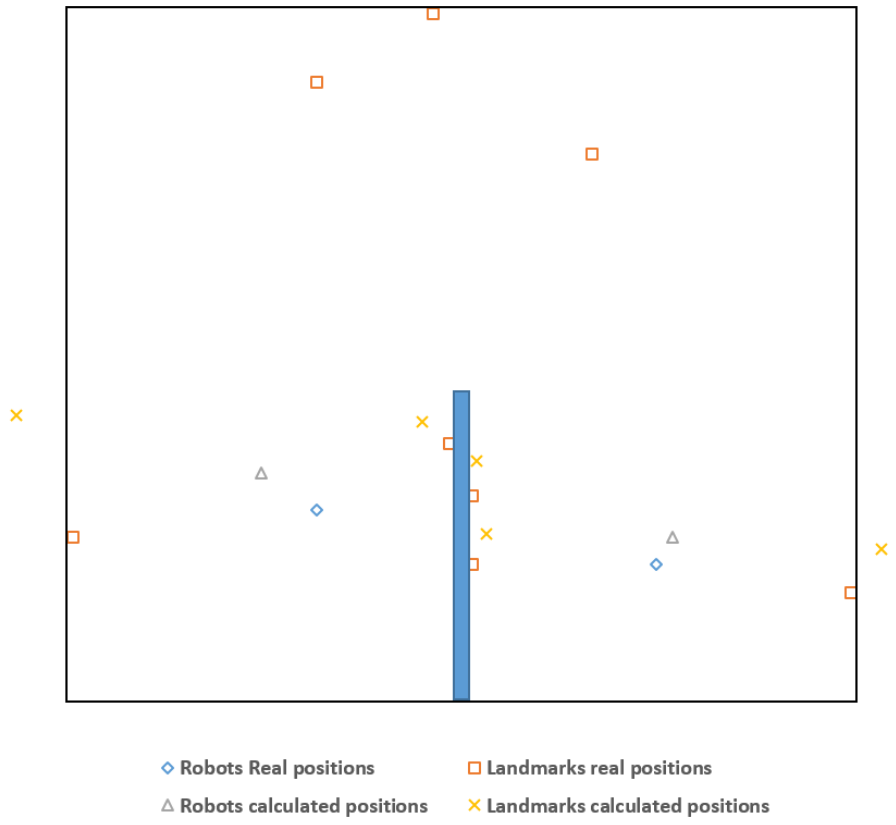

Figure 4.3 *Result of the mapping in the first experiment.*

4.3.2 Second experiment

After the implementation of the first experiment, in the second one it was analyzed the effect of calculate the robots' location with just two known landmarks instead of one. For that purpose, it was designed the experiment with the colocation and the results present in the Table 5.

Table 5 *Results of the second experiment.*

		Real			Calculated		
Type	**ID**	**X(m)**	**Y(m)**	**Orientation (rad)**	**X(m)**	**Y(m)**	**Orientation (rad)**
Robot	bluenao	0.75	0.80	1.57	0.58	0.81	1.50
Robot	orangenao	1.80	0.40	1.57	1.86	0.64	1.65
UnknownLandmark	[107]	0	0.80		-0.21	0.95	
UnknownLandmark	[108]	1.23	0.40		1.28	0.64	
UnknownLandmark	[112]	1.23	0.65		1.25	0.90	
UnknownLandmark	[114]	2.40	0.30		2.51	0.61	
KnownLandmark	[64]	0.80	2.40				
KnownLandmark	[119]	1.60	1.89				

Representing the results of this second experiment, we will obtain the map in the Figure 4.4.

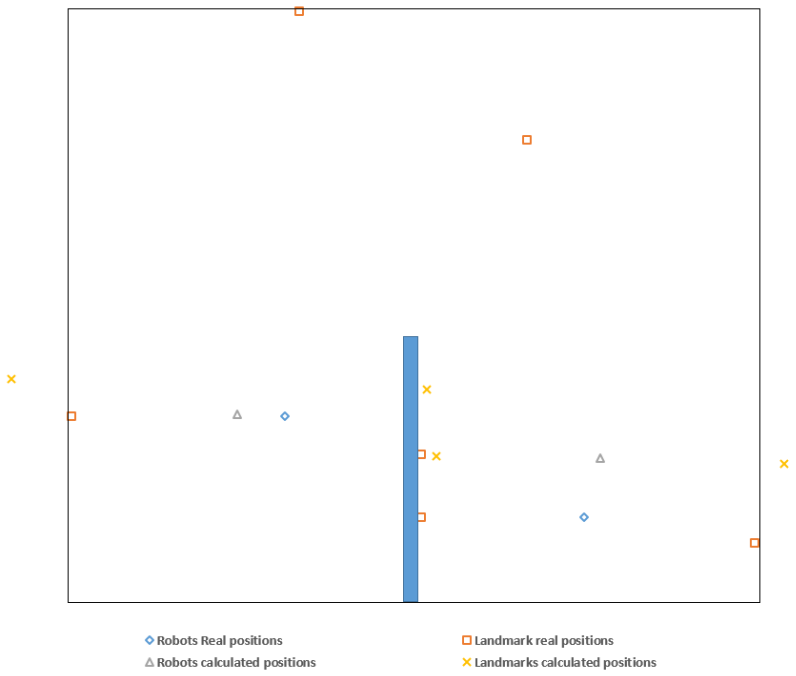

Figure 4.4 *Result of the mapping in the second experiment.*

If we analyze the results of this second experiment, we can see that the error on the calculated location of the robots as well as for the unknown landmarks has grown. Now the absolute averaged error is around 15.1 cm, which is a relative error of 6.3 %. This increase in the error is mainly caused because of the less number of known landmarks available for the robot localization in the map. Furthermore, the error in the orientation calculation of the robots has also been increased, there is an absolute averaged error of 0.14 radians (4.4%).

4.3.3 Third experiment

The aim of the third experiment is to analyze the effect of the distance between the landmarks used for the calculation of the location of the robots. In this experiment, the position of the robots stayed as in the second experiment, but this time the two known landmarks were situated closer between them than in the second one to analyze its effect in the localization of the robot and unknown landmarks, as we can see in the Table 6.

Table 6 *Results of the third experiment.*

		Real			Calculated		
Type	**ID**	**X(m)**	**Y(m)**	**Orientation (rad)**	**X(m)**	**Y(m)**	**Orientation (rad)**
Robot	bluenao	0.75	0.80	1.57	0.45	0.91	1.37
Robot	orangenao	1.80	0.40	1.57	1.73	0.42	1.54
UnknownLandmark	[107]	0.00	0.50		-0.35	1.09	
UnknownLandmark	[84]	1.16	0.84		0.87	0.91	
UnknownLandmark	[114]	2.40	0.30		2.37	0.30	
UnknownLandmark	[112]	1.23	0.65		1.14	0.71	
UnknownLandmark	[108]	1.23	0.40		1.15	0.46	
KnownLandmark	[68]	0.80	2.23				
KnownLandmark	[119]	1.60	2.06				

Representing the results of the third experiment in the map, we would obtain the Figure 4.5.

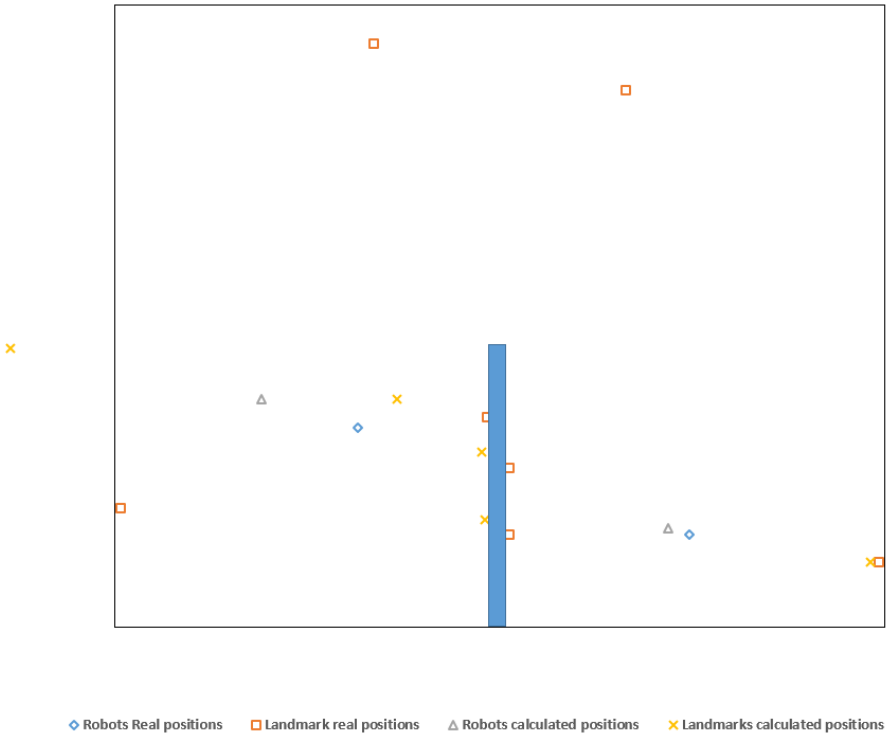

Figure 4.5 *Result of the mapping in the third experiment.*

In this third experiment, we can observe that, in the case of bluenao, the accuracy of the localization of the robot has decreased, mainly caused of the relative distance between the two known landmarks used for the localization of the robots. This can cause that the accuracy of the results will be worse. In the second experiment the relative distance between the known landmarks was 65 cm, while in this experiment it has been reduced to 48cm. It's also important that especially the Y coordinate in both landmarks is very similar, which also causes a rise in the error.

The bad localization of bluenao is also reflected in the posterior calculation of the unknown landmark's position, where we see that the error in the landmarks [84] and [107] has grown significantly.

4.3.4 Fourth experiment

Finally, in the last experiment, the intention is to test if the distance from NAO robots to the known landmarks has an influence in the result of the mapping task. In this fourth experiment, the two robots were situated closer to the X axis, it is, further to the known landmarks, as we can see in the Table 7.

Table 7 *Results of the fourth experiment.*

		Real			Calculated		
Type	**ID**	**X(m)**	**Y(m)**	**Orientation (rad)**	**X(m)**	**Y(m)**	**Orientation (rad)**
Robot	bluenao	0.75	0.80	1.57	0.45	0.91	1.37
Robot	orangenao	1.80	0.40	1.57	1.73	0.42	1.54
UnknownLandmark	[107]	0.00	0.50		-0.35	1.09	
UnknownLandmark	[84]	1.16	0.84		0.87	0.91	
UnknownLandmark	[114]	2.40	0.30		2.37	0.30	
UnknownLandmark	[112]	1.23	0.65		1.14	0.71	
UnknownLandmark	[108]	1.23	0.40		1.15	0.46	
KnownLandmark	[68]	0.80	2.23				
KnownLandmark	[119]	1.60	2.06				

In the Figure 4.6 it has been represented these results in the map.

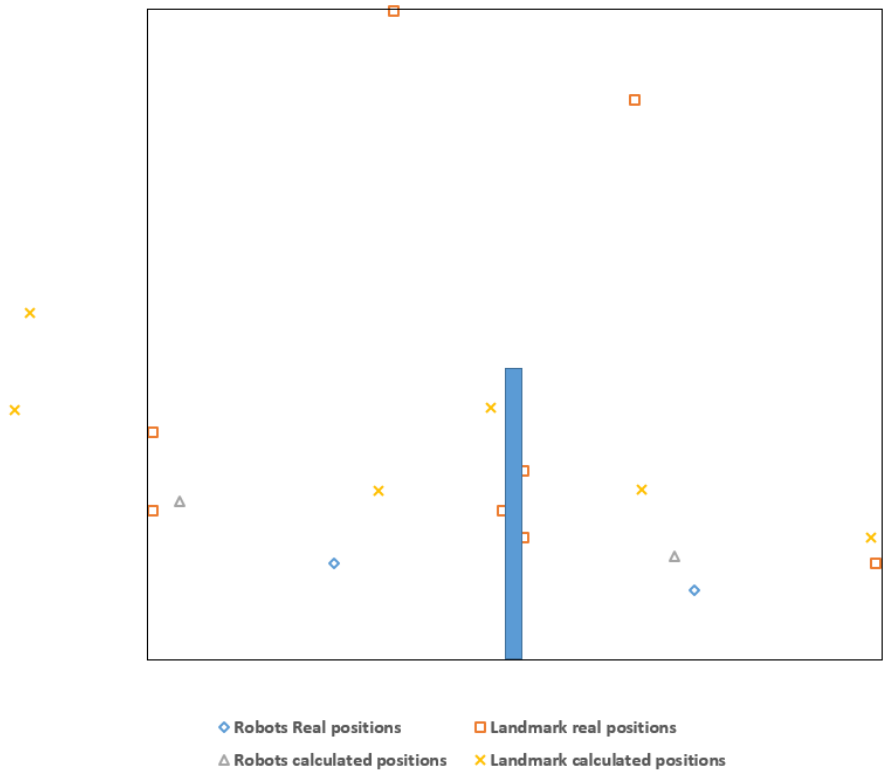

Figure 4.6 *Result of the mapping in the fourth experiment.*

Analyzing the results of this last experiment, we see that situate the robots far from the known landmarks has a big impact on the results. We see that now the absolute averaged error has raised to 20.7 cm, a relative error of 8.6 %.

This is caused because the measurement of the distance to the landmark from part of NAO robot is worse when the landmark is farther, and a bad localization of the robot in the map, position and specially orientation (34.3 % of relative error in this case) will have later a big influence in the positioning of the unknown landmarks, as we can see in this experiment.

4.4 Final conclusions

To sum up all the conclusions obtained from the experiments, we can observe the Table 8 with a summary of the four experiments.

Table 8 *Summary of the experiments' results.*

Experiment	Comments	Relative averaged error	Robots position relative error	Robot orientation relative error
1	Three known landmarks with a good separation between them	5.3 %	4.7 %	2.5%
2	Two known landmarks with a good separation between them	6.3 %	4.8 %	4.4 %
3	Two known landmarks closer between them	6.3 %	5.2 %	3.6 %
4	Two known landmarks with a good separation between them and robots located far from them	8.6 %	9.7 %	34.3 %

Finally, these are the overall conclusions we can obtain from the experiments:

- The localization of the robot in the map is a crucial aspect of the mapping task, since a bad pose calculation (position and orientation) of the robots will extend the error to the unknown landmark position in the map.

- When we add more known landmarks to the map it has a good effect in the results, firstly because the robot localization accuracy will rise and therefore the unknown landmarks mapping accuracy, and secondly because if some robot doesn't detect one of the known landmark, it can still be located in the map with the others.
- The closer the known landmarks are in the map, especially in the same axis, the worse will be the localization of the robots and, therefore, the global results of the mapping task. In the experiment 3 it was explained that a closer location of the landmarks may cause a bad localization of one of the robots and create a distortion in the results.
- If the distance from the robots and the known landmarks is big, the measure of this distance of the robot will be bad and therefore the overall results of the mapping.

In conclusion, the results obtained for the code created have been positive, the overall error in most of the cases is not very significant and, which is also important, the distance between NAO robots is accurate, obtained without the need of have a direct visual contact between them.

5 CONCLUSION AND FUTURE WORK

In this chapter, it will be exposed the conclusion of the thesis, as well as a possible future work for the project.

5.1 Conclusion of the project

In this document, it has been described the procedure to develop an application for NAO robot to create a map of a partially known area in collaboration with other robots. First of all, the objectives and the state of the art was described, as well as a brief explanation of the tools used. Then, the methodology to develop the program and the solutions for some problems that appeared during this development were presented and, finally, the experiments carried out to test the code of the program and see the different drivers that may change the results were explained as well.

After the development of the program and the understanding of the aspects that involves working with humanoid robots, it can be said that the project has been successful. The code created is able to make some robots collaborate and communicate between them sharing their own information in the program to create the map of the area where they are located.

The objectives that were described in the first chapter of this document have been completely achieved and, as a result, the personal knowledge has been improved, as well as a contribution to the department's research has been made.

The code developed during the thesis has been tested with different experiments and the results of them have been satisfactory. The program is able to make two robots work together and collaborate to do the mapping of a partially-known area, situate themselves and other unknown objects (represented with landmarks) in the map with a good accuracy.

Personally, it has been a clear improvement of the knowledge in the area of humanoid robots, how to program them and the understanding of the challenges in the field of mapping in autonomous mobile robots. Working with NAO has been a great opportunity to understand the operation of these robots and the possibilities present in them.

5.2 Possible future work

Regarding the possible future work and applications of this project, it can be found a lot of opportunities for future developments.

First of all, the code used for the project could still be more efficiently programmed, using concepts as dynamic memory. Also, the results of the mapping could be improved as well, for example constraining the results to the area that we're working in, so there are no results out of range. Regarding the content of the experiments, a good idea would have been to make both robots map the same unknown landmark and see the difference between the locations obtained of both of them.

Another future applications that could be found for this project could be the implementation of movement, once the map is already known, to a concrete point of the map, or, for instance, implement a code that takes into account obstacles in the environment, not just the limits of them, like we implemented in this project. The future applications for collaboration between humanoid robots for the mapping task are abundant.

Furthermore, there could be used another methods for the mapping task of the robot, not just with naomarks, but also with another type of marks like QR codes, or even with real world elements, even though adapt the code from landmarks to real objects is a difficult task because of the range of possible objects that could need to be located.

REFERENCES

[1] Ian Sample, "The Guardian," 2016. [Online]. Available: https://www.theguardian.com/science/2016/jan/19/stephen-hawking-warns-threats-to-humans-science-technology-bbc-reith-lecture.

[2] S. S. a. G. Mooney, "Urbanization, mortality and the standard of living debate," *The Economic History Review,* p. 29, 1998.

[3] R. S. M. C. M. R. and D. S. , Autonomous Mobile Robots, ETH Zürich, 2013.

[4] B. Dynamics, "Spot description," 2017. [Online]. Available: https://www.bostondynamics.com/spot.

[5] Kudan, "An Introduction to Simultaneous Localisation and Mapping," 2016. [Online]. Available: https://www.kudan.eu/kudan-news/an-introduction-to-slam/.

[6] Aethon, "Self-driving robot navigation methodology explained.," 2016. [Online]. Available: http://www.aethon.com/our-navigation-methodology-explained/.

[7] Aldebaran, "Unveiling of NAO Evolution: a stronger robot and a more comprehensive operating system," 2014. [Online]. Available: https://www.ald.softbankrobotics.com/sites/aldebaran/files/press-releases/cp_nao_evolution_en_def.pdf.

[8] Aldebaran, "Who is NAO?," 2016. [Online]. Available: https://www.ald.softbankrobotics.com/en/robots/nao.

[9] Aldebaran, "NAO Documentation," 2016. [Online]. Available: http://doc.aldebaran.com/2-1/home_nao.html.

[10] Qt, "Qt APIs & Tools, Libraries and Qt Creator IDE," 2017. [Online]. Available: https://www.qt.io/qt-features-libraries-apis-tools-and-ide/.

[11] M. Heidarysafa, MSc: Heuristic localization and mapping for active sensing with humanoid robot NAO, 2015, p. 69.

[12] U. C. I. d. M. Department of Systems and Automatic Engineering, Kinematic control, industrial robotics course, 2011.

[13] "Eigen 3.3.4 documentation," 2017. [Online].

[14] A. Uwaoma, MSc: On particle filter localization and mapping for nao robbot, 2015, p. 86.

[15] W. K. Jan Figat, NAO-mark vs QR-code recognition by Nao robotvision, Institute of Control and Computation Engineering, Warsaw University of Technology, 2016.

[16] A. Marin-Hernandez, NAO Developers Tutorial: with Naoqi C++ SDK, Department of Artificial Intelligence, Universidad Veracruzana, 2014.

[17] J. A. F. Madrigal and J. L. B. Claraco, Simultaneous Localization and Mapping for Mobile Robots, Universidad de Málaga, 2013.

APPENDIX A: C++ CODE

ROBOT CLASS

Robot.h

```
#ifndef ROBOT_ROBOT_H
#define ROBOT_ROBOT_H

#include <boost/shared_ptr.hpp>
#include <alcommon/almodule.h>
#include <string>

#include <alproxies/almemoryproxy.h>
#include <alproxies/altexttospeechproxy.h>
#include <alproxies/almotionproxy.h>
#include <althread/almutex.h>
#include <almath/types/altransform.h>


#define MAXLANDMARKS 10

namespace AL
{
  class ALBroker;
}

class Robot : public AL::ALModule
{
  public:

    Robot(boost::shared_ptr<AL::ALBroker> broker, const std::string &name);

    virtual ~Robot();

    virtual void init();

    void configure();

    void onLandmarkDetected();

    void scan();

    void calculatepositions();

    void fixpositions();

    AL::ALValue getMarkerID(int n, int i);

    int getLandmarksDetected(int n);

    int getTurnsCompleted();

    float getx(int n, int i), gety(int n, int i);

    float getxav(int n), getyav(int n);

    AL::ALValue getMarkerIDfixed(int n);

    int getLandmarksDetectedFixed();

    void exporttofile(std::string nameoffile);

    //Auxiliary methods to display data from the robot
```

```
    void showLandmarksDetected(boost::shared_ptr<AL::ALProxy> nao, std::string
name);

    void showIdsDetected(boost::shared_ptr<AL::ALProxy> nao, std::string
name);

    void showPositionsDetected(boost::shared_ptr<AL::ALProxy> nao, std::string
name);

    void showAveragedPositiondsDetected(boost::shared_ptr<AL::ALProxy> nao,
std::string name);


  private:

    AL::ALMemoryProxy MemoryProxy;
    AL::ALTextToSpeechProxy SpeechProxy;
    AL::ALMotionProxy MotionProxy;
    boost::shared_ptr<AL::ALMutex> fCallbackMutex;
    std::string name;

    AL::ALValue MarkersID[3][MAXLANDMARKS];
    AL::ALValue MarkersIDfixed[MAXLANDMARKS];
    int LandmarksDetected[3];
    int LandmarksDetectedfixed;
    int TurnsCompleted;
    float x[3][MAXLANDMARKS], y[3][MAXLANDMARKS];
    float xav[MAXLANDMARKS], yav[MAXLANDMARKS];

    float wzCamera[3][MAXLANDMARKS];
    float wyCamera[3][MAXLANDMARKS];
    float angularsize[3][MAXLANDMARKS];
    std::vector<float> results[3][MAXLANDMARKS];

    void say(std::string whattosay);
};

#endif  // ROBOT_ROBOT_H
```

Robot.cpp

```
#include "robot.h"

#include <alvalue/alvalue.h>
#include <alcommon/alproxy.h>
#include <alcommon/albroker.h>
#include <qi/log.hpp>
#include <althread/alcriticalsection.h>


#include <iostream>
#include <fstream>
#include <iomanip>

#define LANDMARKSIZE 0.155

#define SPEED 0.3f
#define ACCURACY 0.007f //Head positioning accuracy (influencing also in the
speed)
#define TIMECONFIGPITCH 0.5f
#define TIMECONFIGYAW 1.5f
#define TIMESCAN 10.0f



Robot::Robot(boost::shared_ptr<AL::ALBroker> broker, const std::string& name)
: AL::ALModule(broker, name), fCallbackMutex(AL::ALMutex::createALMutex()),
```

```
    name(name) {

  qiLogInfo("Robot.module") << name << " created succesfuly."<<std::endl;

  //Binding methods so they can be called in the main

  setModuleDescription("This module presents how the robot behave to detect
landmarks using a callback method.");
  functionName("configure", getName(), "Method that connect to the robot prox-
ies, subscribe to event and position the head");
  BIND_METHOD(Robot::configure);

  functionName("getMarkerID", getName(), "Method to get the id of a landmark
detected");
  addParam("n", "The number of the turn to get the id");
  addParam("i", "The number of the landmark to get the id");
  setReturn("MarkerID", "The id requested");
  BIND_METHOD(Robot::getMarkerID);

  functionName("getLandmarksDetected", getName(), "Method to get the total
number of landmarks detected");
  addParam("n", "The number of the turn to get the landmarksdetected");
  setReturn("LandmarksDetected", "Total number of landmarks detected");
  BIND_METHOD(Robot::getLandmarksDetected);

  functionName("getTurnscompleted", getName(), "Method to get the total number
of turns completed");
  setReturn("Turnscompleted", "Total number of turns completed");
  BIND_METHOD(Robot::getTurnsCompleted);

  functionName("getx", getName(), "Method to get the x position");
  addParam("n", "The number of the turn to get the position");
  addParam("i", "The number of the landmark to get the position");
  setReturn("x", "The x position requested");
  BIND_METHOD(Robot::getx);

  functionName("gety", getName(), "Method to get the y position");
  addParam("n", "The number of the turn to get the position");
  addParam("i", "The number of the landmark to get the position");
  setReturn("y", "The y position requested");
  BIND_METHOD(Robot::gety);

  functionName("getxav", getName(), "Method to get the x position averaged of
a Landmark");
  addParam("n", "The number of the landmark to get the position");
  setReturn("x", "The x averaged position requested");
  BIND_METHOD(Robot::getxav);

  functionName("getyav", getName(), "Method to get the y position averaged");
  addParam("n", "The number of the landmark to get the position");
  setReturn("yav", "The y position averaged requested");
  BIND_METHOD(Robot::getyav);

  functionName("getMarkerIDfixed", getName(), "Method to get the fixed Mark-
erID");
  addParam("n", "The number of the landmark to get the id fixed");
  setReturn("MarkerIDfixed", "The fixed MarkerID requested");
  BIND_METHOD(Robot::getMarkerIDfixed);

  functionName("getLandmarksDetectedFixed", getName(), "Method to get the to-
tal number of landmarks detected fixed");
  setReturn("landmarksdetectedfixed", "Total number of landmarks detected
fixed");
  BIND_METHOD(Robot::getLandmarksDetectedFixed);

  functionName("scan", getName(), "Method called to move the head 360 degrees
from left to right 3 times");
  BIND_METHOD(Robot::scan);
```

```
  functionName("onLandmarkDetected", getName(), "Method called when the robot
detect a landmark.");
  BIND_METHOD(Robot::onLandmarkDetected);

  functionName("calculatepositions", getName(), "Method called to calculate
the position of every landmark on every turn with the transform");
  BIND_METHOD(Robot::calculatepositions);

  functionName("fixpositions", getName(), "Method called to calculate the fi-
nal position of every landmark using average");
  BIND_METHOD(Robot::fixpositions);

  functionName("exporttofile", getName(), "Method that export the data to a
file.");
  addParam("nameoffile", "The name of the file to be exported");
  BIND_METHOD(Robot::exporttofile);

  functionName("say", getName(), "Method that says something.");
  addParam("whattosay", "What is going to be said by the robot");
  BIND_METHOD(Robot::say);

 }



Robot::~Robot() {
  MemoryProxy.unsubscribeToEvent("LandmarkDetected", name);     //When the
program is over, the robot has to unsuscribe to the landmark detection
}



void Robot::init() {
}



void Robot::configure(){

    MemoryProxy = AL::ALMemoryProxy(getParentBroker());
    SpeechProxy = AL::ALTextToSpeechProxy(getParentBroker());
    MotionProxy = AL::ALMotionProxy(getParentBroker());


    //Configuring the head position
    MotionProxy.setStiffnesses("Head", 1.0f);

    MotionProxy.angleInterpolation("HeadPitch", 0.0f, TIMECONFIGPITCH, true);

    if(MotionProxy.getAngles("HeadYaw",0)[0]>0){                //if it's
closer to left limit

    MotionProxy.angleInterpolation("HeadYaw", 1.57f, TIMECONFIGYAW, true);
//move left

    }
    else{                        //if it's closer to right limit

    MotionProxy.angleInterpolation("HeadYaw", -1.57f, TIMECONFIGYAW, true);
//move right

    }

   MotionProxy.setStiffnesses("Head", 0.0f);

   //Reset all the variables
   TurnsCompleted = 0;
   for (int i=0;i<3;i++){
       for(int n=0;n<MAXLANDMARKS;n++){
           MarkersID[i][n]=0;
```

```
            LandmarksDetected[i]=0;
            x[i][n], y[i][n]=0;
            xav[n], yav[n]=0;
            MarkersIDfixed[n]=0;
            wzCamera[i][n]=0;
            wyCamera[i][n]=0;
            angularsize[i][n]=0;
            results[i][n]=std::vector<float>(0);
        }
    }

   //Once the head is configured, the subscription to the event "Landmark-
Detected" has to be realized
   MemoryProxy.subscribeToEvent("LandmarkDetected", name, "onLandmark-
Detected");
}


AL::ALValue Robot::getMarkerID(int n, int i){return MarkersID[n][i];}


int Robot::getLandmarksDetected(int n){return LandmarksDetected[n];}


int Robot::getTurnsCompleted(){return TurnsCompleted;}


float Robot::getx(int n, int i){return x[n][i];}


float Robot::gety(int n, int i){return y[n][i];}


float Robot::getxav(int n){return xav[n];}


float Robot::getyav(int n){return yav[n];}


AL::ALValue Robot::getMarkerIDfixed(int n){return MarkersIDfixed[n];}


int Robot::getLandmarksDetectedFixed(){return LandmarksDetectedfixed;}


void Robot::scan(){

    for(TurnsCompleted=0;TurnsCompleted<3;TurnsCompleted++){

        MotionProxy.setStiffnesses("Head", 1.0f);

       if(MotionProxy.getAngles("HeadYaw",0)[0]>0){                //if it's
in the left limit

              MotionProxy.angleInterpolation("HeadYaw", -1.57f, TIMESCAN,
true);       //move right

       }
       else{                                                       //if it's
in the right limit
```

```
                MotionProxy.angleInterpolation("HeadYaw", 1.57f, TIMESCAN,
true);       //move left

       }

        MotionProxy.setStiffnesses("Head", 0.0f);
    }
}



void Robot::onLandmarkDetected() {

  AL::ALCriticalSection section(fCallbackMutex);

   AL::ALValue data = MemoryProxy.getData("LandmarkDetected");

  if(data.getSize()==0){return;}        //if there is no landmark on objective
(because the head is moving)

  if(data[1][0][1].toString()=="0"){return;}

  for(int i=0; i<LandmarksDetected[TurnsCompleted]; i++){   //check if we al-
ready detected this landmark in this turn (repeated)
      if(data[1][0][1] == MarkersID[TurnsCompleted][i]){
          return;
      }
  }

  MarkersID[TurnsCompleted][LandmarksDetected[TurnsCompleted]] =
data[1][0][1];
  wzCamera[TurnsCompleted][LandmarksDetected[TurnsCompleted]] =
data[1][0][0][1];
  wyCamera[TurnsCompleted][LandmarksDetected[TurnsCompleted]] =
data[1][0][0][2];
  angularsize[TurnsCompleted][LandmarksDetected[TurnsCompleted]] =
data[1][0][0][3];
  results[TurnsCompleted][LandmarksDetected[TurnsCompleted]] = Mo-
tionProxy.getTransform("CameraTop", 2, true);

  LandmarksDetected[TurnsCompleted]++;
}



void Robot::calculatepositions(){

    for(int n=0;n<3;n++){
        for(int i=0;i<LandmarksDetected[n];i++){
            float disCameratoLandmark = LANDMARKSIZE / (2 * tan(angular-
size[n][i]/2));
            AL::Math::Transform robotToCamera(results[n][i]);
            AL::Math::Transform cameraToLandmarkRotTrans = AL::Math::Trans-
form::from3DRotation(0, wyCamera[n][i], wzCamera[n][i]);
            AL::Math::Transform cameraToLandmarkTranslTrans = AL::Math::Trans-
form::fromPosition(disCameratoLandmark,0,0);
            AL::Math::Transform robotToLandmark = robotToCamera * cameraTo-
LandmarkRotTrans * cameraToLandmarkTranslTrans;

            x[n][i] = robotToLandmark.r1_c4;
            y[n][i] = robotToLandmark.r2_c4;
        }
    }
}



void Robot::fixpositions(){
```

```
    //Once we have calculated the positions, the landmarks that has just been
detected in one turn have to be deleted
    //In this function the average between the data of the three turns is cal-
culated as well

    int index = 0;

    for(int i=0;i<LandmarksDetected[0];i++){

        for(int j=0;j<LandmarksDetected[1];j++){

            if(MarkersID[0][i]==MarkersID[1][j]){

                for(int k=0;k<LandmarksDetected[2];k++){

                    if(MarkersID[1][j]==MarkersID[2][k]){

                        MarkersIDfixed[index]=MarkersID[0][i];
                        xav[index]= (x[0][i]+x[1][j]+x[2][k])/3;
                        yav[index]= (y[0][i]+y[1][j]+y[2][k])/3;
                        index++;

                        if(MarkersIDfixed[index]==AL::ALValue("0")){
                            index--;
                        }
                    }
                }
            }
        }
    }

    LandmarksDetectedfixed=index;

}



void Robot::exporttofile(std::string nameoffile){

    std::ofstream file;
    file.open(nameoffile.c_str());

    file << "ID,X,Y\n";     //Columns separated by commas

    for(int i=0;i<LandmarksDetectedfixed;i++){
        file << MarkersIDfixed[i] << std::setprecision(2)<< ","<<xav[i] << ","
<< yav[i] <<"\n";
    }

    file.close();
}



void Robot::say(std::string whattosay){
    SpeechProxy.say(whattosay);
}


//Auxiliar methods to display data from the robot


void showLandmarksDetected(boost::shared_ptr<AL::ALProxy> nao, std::string
name){
    qiLogInfo("Main") << "Number of Landmarks Detected by " << name << ": " <<
nao->call<int>("getLandmarksDetected",0) << " / " << nao->call<int>("getLand-
marksDetected",1) << " / " << nao->call<int>("getLandmarksDetected",2) <<
std::endl;
}
```

```
void showIdsDetected(boost::shared_ptr<AL::ALProxy> nao, std::string name){
    qiLogInfo("Main") << name << " detected this Landmarks: "<< nao-
>call<AL::ALValue>("getMarkerID", 0,0) << " " << nao->call<AL::AL-
Value>("getMarkerID", 0,1) << " " << nao->call<AL::ALValue>("getMarkerID",
0,2)<< " "
                      << nao->call<AL::ALValue>("getMarkerID", 0,3)<< " " <<
nao->call<AL::ALValue>("getMarkerID", 0,4)<< " " << nao->call<AL::AL-
Value>("getMarkerID", 0,5)<< " "
                      << nao->call<AL::ALValue>("getMarkerID", 0,6)<< " " <<
nao->call<AL::ALValue>("getMarkerID", 0,7)<< " " << nao->call<AL::AL-
Value>("getMarkerID", 0,8)<< " "
                      << nao->call<AL::ALValue>("getMarkerID", 0,9)<< " / " <<
nao->call<AL::ALValue>("getMarkerID", 1,0) << " " << nao->call<AL::AL-
Value>("getMarkerID", 1,1) << " " << nao->call<AL::ALValue>("getMarkerID",
1,2)<< " "
                      << nao->call<AL::ALValue>("getMarkerID", 1,3)<< " " <<
nao->call<AL::ALValue>("getMarkerID", 1,4)<< " " << nao->call<AL::AL-
Value>("getMarkerID", 1,5)<< " "
                      << nao->call<AL::ALValue>("getMarkerID", 1,6)<< " " <<
nao->call<AL::ALValue>("getMarkerID", 1,7)<< " " << nao->call<AL::AL-
Value>("getMarkerID", 1,8)<< " "
                      << nao->call<AL::ALValue>("getMarkerID", 1,9)<< " / " <<
nao->call<AL::ALValue>("getMarkerID", 2,0) << " " << nao->call<AL::AL-
Value>("getMarkerID", 2,1) << " " << nao->call<AL::ALValue>("getMarkerID",
2,2)<< " "
                      << nao->call<AL::ALValue>("getMarkerID", 2,3)<< " " <<
nao->call<AL::ALValue>("getMarkerID", 2,4)<< " " << nao->call<AL::AL-
Value>("getMarkerID", 2,5)<< " "
                      << nao->call<AL::ALValue>("getMarkerID", 2,6)<< " " <<
nao->call<AL::ALValue>("getMarkerID", 2,7)<< " " << nao->call<AL::AL-
Value>("getMarkerID", 2,8)<< " "
                      << nao->call<AL::ALValue>("getMarkerID",
2,9)<<std::endl;
}



void showPositionsDetected(boost::shared_ptr<AL::ALProxy> nao, std::string
name){

    qiLogInfo("Main") << name << " detected this Xs positions: "<< nao-
>call<AL::ALValue>("getx", 0,0) << " " << nao->call<AL::ALValue>("getx", 0,1)
<< " " << nao->call<AL::ALValue>("getx", 0,2)<< " "
                      << nao->call<AL::ALValue>("getx", 0,3)<< " " << nao-
>call<AL::ALValue>("getx", 0,4)<< " " << nao->call<AL::ALValue>("getx", 0,5)<<
" "
                      << nao->call<AL::ALValue>("getx", 0,6)<< " " << nao-
>call<AL::ALValue>("getx", 0,7)<< " " << nao->call<AL::ALValue>("getx", 0,8)<<
" "
                      << nao->call<AL::ALValue>("getx", 0,9)<<" / " << nao-
>call<AL::ALValue>("getx", 1,0) << " " << nao->call<AL::ALValue>("getx", 1,1)
<< " " << nao->call<AL::ALValue>("getx", 1,2)<< " "
                      << nao->call<AL::ALValue>("getx", 1,3)<< " " << nao-
>call<AL::ALValue>("getx", 1,4)<< " " << nao->call<AL::ALValue>("getx", 1,5)<<
" "
                      << nao->call<AL::ALValue>("getx", 1,6)<< " " << nao-
>call<AL::ALValue>("getx", 1,7)<< " " << nao->call<AL::ALValue>("getx", 1,8)<<
" "
                      << nao->call<AL::ALValue>("getx", 1,9)<<  " / " << nao-
>call<AL::ALValue>("getx", 2,0) << " " << nao->call<AL::ALValue>("getx", 2,1)
<< " " << nao->call<AL::ALValue>("getx", 2,2)<< " "
                      << nao->call<AL::ALValue>("getx", 2,3)<< " " << nao-
>call<AL::ALValue>("getx", 2,4)<< " " << nao->call<AL::ALValue>("getx", 2,5)<<
" "
                      << nao->call<AL::ALValue>("getx", 2,6)<< " " << nao-
>call<AL::ALValue>("getx", 2,7)<< " " << nao->call<AL::ALValue>("getx", 2,8)<<
" "
```

```
                        << nao->call<AL::ALValue>("getx", 2,9)<< " "
<<std::endl;

    qiLogInfo("Main") << name << " detected this Ys positions: "<< nao-
>call<AL::ALValue>("gety", 0,0) << " " << nao->call<AL::ALValue>("gety", 0,1)
<< " " << nao->call<AL::ALValue>("gety", 0,2)<< " "
                        << nao->call<AL::ALValue>("gety", 0,3)<< " " << nao-
>call<AL::ALValue>("gety", 0,4)<< " " << nao->call<AL::ALValue>("gety", 0,5)<<
" "
                        << nao->call<AL::ALValue>("gety", 0,6)<< " " << nao-
>call<AL::ALValue>("gety", 0,7)<< " " << nao->call<AL::ALValue>("gety", 0,8)<<
" "
                        << nao->call<AL::ALValue>("gety", 0,9)<< " " <<  " / "
<< nao->call<AL::ALValue>("gety", 1,0) << " " << nao->call<AL::AL-
Value>("gety", 1,1) << " " << nao->call<AL::ALValue>("gety", 1,2)<< " "
                        << nao->call<AL::ALValue>("gety", 1,3)<< " " << nao-
>call<AL::ALValue>("gety", 1,4)<< " " << nao->call<AL::ALValue>("gety", 1,5)<<
" "
                        << nao->call<AL::ALValue>("gety", 1,6)<< " " << nao-
>call<AL::ALValue>("gety", 1,7)<< " " << nao->call<AL::ALValue>("gety", 1,8)<<
" "
                        << nao->call<AL::ALValue>("gety", 1,9)<< " / " << nao-
>call<AL::ALValue>("gety", 2,0) << " " << nao->call<AL::ALValue>("gety", 2,1)
<< " " << nao->call<AL::ALValue>("gety", 2,2)<< " "
                        << nao->call<AL::ALValue>("gety", 2,3)<< " " << nao-
>call<AL::ALValue>("gety", 2,4)<< " " << nao->call<AL::ALValue>("gety", 2,5)<<
" "
                        << nao->call<AL::ALValue>("gety", 2,6)<< " " << nao-
>call<AL::ALValue>("gety", 2,7)<< " " << nao->call<AL::ALValue>("gety", 2,8)<<
" "
                        << nao->call<AL::ALValue>("gety", 2,9)<< " "
<<std::endl;
}



void showAveragedPositiondsDetected(boost::shared_ptr<AL::ALProxy> nao,
std::string name){

    qiLogInfo("Main") << name << " detected " << nao->call<int>("getLand-
marksDetectedFixed") << " landmarks fixed. Numbers: "<< nao->call<AL::AL-
Value>("getMarkerIDfixed", 0) << " "
                        << nao->call<AL::ALValue>("getMarkerIDfixed", 1) << " "
<< nao->call<AL::ALValue>("getMarkerIDfixed", 2)<< " "
                        << nao->call<AL::ALValue>("getMarkerIDfixed", 3)<< " "
<< nao->call<AL::ALValue>("getMarkerIDfixed", 4)<< " " << nao->call<AL::AL-
Value>("getMarkerIDfixed", 5)<< " "
                        << nao->call<AL::ALValue>("getMarkerIDfixed", 6)<< " "
<< nao->call<AL::ALValue>("getMarkerIDfixed", 7)<< " " << nao->call<AL::AL-
Value>("getMarkerIDfixed", 8)<< " "
                        << nao->call<AL::ALValue>("getMarkerIDfixed",
9)<<std::endl;

    qiLogInfo("Main") << name << " detected this Xs averaged positions: "<<
nao->call<AL::ALValue>("getxav", 0) << " " << nao->call<AL::ALValue>("getxav",
1) << " " << nao->call<AL::ALValue>("getxav", 2)<< " "
                        << nao->call<AL::ALValue>("getxav", 3)<< " " << nao-
>call<AL::ALValue>("getxav", 4)<< " " << nao->call<AL::ALValue>("getxav", 5)<<
" "
                        << nao->call<AL::ALValue>("getxav", 6)<< " " << nao-
>call<AL::ALValue>("getxav", 7)<< " " << nao->call<AL::ALValue>("getxav", 8)<<
" "
                        << nao->call<AL::ALValue>("getxav", 9)<< " "
<<std::endl;

    qiLogInfo("Main") << name << " detected this Ys averaged positions: "<<
nao->call<AL::ALValue>("getyav", 0) << " " << nao->call<AL::ALValue>("getyav",
1) << " " << nao->call<AL::ALValue>("getyav", 2)<< " "
```

```
                        << nao->call<AL::ALValue>("getyav", 3)<< " " << nao-
>call<AL::ALValue>("getyav", 4)<< " " << nao->call<AL::ALValue>("getyav", 5)<<
" "
                        << nao->call<AL::ALValue>("getyav", 6)<< " " << nao-
>call<AL::ALValue>("getyav", 7)<< " " << nao->call<AL::ALValue>("getyav", 8)<<
" "
                        << nao->call<AL::ALValue>("getyav", 9)<< " "
<<std::endl;

}
```

MAP CLASS

Map.h

```
#ifndef ROBOT_MAP_H
#define ROBOT_MAP_H

#include <boost/shared_ptr.hpp>
#include <alcommon/almodule.h>

#define MAXROBOTS 5
#define MAXKNOWNLANDMARKS 5
#define MAXUNKNOWNLANDMARKS 20

struct RobotInMap{
    std::string name;
    float x, y, orientation;
};

struct Landmark{
    AL::ALValue name;
    float x, y;
};


class Map {
  public:

    Map();

    virtual ~Map();

    void initialize();

    RobotInMap getrobotdata(int i);

    Landmark getknownlandmarkdata(int i);

    Landmark getunknownlandmarkdata(int i);

    int gettotalrobots(), gettotalknownlandmarks(), gettotalunknownland-
marks();

    void addknownlandmark(AL::ALValue id, float x, float y);

    void addrobot(boost::shared_ptr<AL::ALProxy> nao, std::string name);

    void addunknownlandmarks(boost::shared_ptr<AL::ALProxy> nao, std::string
name);

    void exporttofile(std::string nameoffile);

    void saydistancetorobot(boost::shared_ptr<AL::ALProxy> nao, std::string
namefrom, std::string nameto);

    //Auxiliary methods to display the robots, knownlandmarks and unknownland-
marks
```

```
    void showrobots();

    void showknownlandmarks();

    void showunknownlandmarks();


  private:

RobotInMap robots[MAXROBOTS];

Landmark knownLandmarks[MAXKNOWNLANDMARKS];

Landmark unknownLandmarks[MAXUNKNOWNLANDMARKS];

int totalRobots, totalKnownLandmarks, totalUnknownLandmarks;

};


#endif  // ROBOT_MAP_H
```

Map.cpp

```
#include "map.h"

#include <alvalue/alvalue.h>
#include <alcommon/alproxy.h>
#include <alcommon/albroker.h>
#include <qi/log.hpp>
#include "/home/sergio/Desktop/SDKfolder/naoqi-sdk/doc/dev/cpp/examples/ro-
bot/Eigen/LU"         //To include Eigen library (change for the folder where
Eigen library actually is)
#include "/home/sergio/Desktop/SDKfolder/naoqi-sdk/doc/dev/cpp/examples/ro-
bot/Eigen/Dense"
#include <math.h>


#include <iostream>
#include <fstream>
#include <iomanip>

#define PI 3.1415926535897f



Map::Map() {

}



Map::~Map() {

}



void Map::initialize(){

    for(int i=0; i<MAXROBOTS; i++){
        robots[i].name="0";
        robots[i].orientation=0;
        robots[i].x=0;
        robots[i].y=0;
    }
```

```
    for(int i=0; i<MAXKNOWNLANDMARKS;i++){
        knownLandmarks[i].name="0";
        knownLandmarks[i].x=0;
        knownLandmarks[i].y=0;
    }

    for(int i=0; i<MAXUNKNOWNLANDMARKS;i++){
        unknownLandmarks[i].name="0";
        unknownLandmarks[i].x=0;
        unknownLandmarks[i].y=0;
    }

    totalKnownLandmarks=0;
    totalRobots=0;
    totalUnknownLandmarks = 0;

}



RobotInMap Map::getrobotdata(int i){return robots[i];}

Landmark Map::getknownlandmarkdata(int i){return knownLandmarks[i];}

Landmark Map::getunknownlandmarkdata(int i){return unknownLandmarks[i];}

int Map::gettotalrobots(){return totalRobots;}

int Map::gettotalknownlandmarks(){return totalKnownLandmarks;}

int Map::gettotalunknownlandmarks(){return totalUnknownLandmarks;}


void Map::addknownlandmark(AL::ALValue id, float x, float y){

    knownLandmarks[totalKnownLandmarks].name=id;
    knownLandmarks[totalKnownLandmarks].x=x;
    knownLandmarks[totalKnownLandmarks].y=y;

    totalKnownLandmarks++;
}



void Map::addrobot(boost::shared_ptr<AL::ALProxy> nao, std::string name){

    float relativepos[MAXKNOWNLANDMARKS][2], known[MAXKNOWNLANDMARKS][2];
//Looking for the knownlandmarks that the robot detected and saving its meas-
urements
    AL::ALValue knownnames[MAXKNOWNLANDMARKS];
    int index=0;
    for(int i=0; i<MAXKNOWNLANDMARKS; i++){
        for(int j=0;j<nao->call<int>("getLandmarksDetectedFixed");j++){
            if(knownLandmarks[i].name!=AL::ALValue("0") && knownLand-
marks[i].name==nao->call<AL::ALValue>("getMarkerIDfixed", j).toString()){
                relativepos[index][0]=nao->call<AL::ALValue>("getxav", j);
                relativepos[index][1]=nao->call<AL::ALValue>("getyav", j);
                known[index][0]=knownLandmarks[i].x;
                known[index][1]=knownLandmarks[i].y;
                knownnames[index]=knownLandmarks[i].name;
                index++;
            }
        }
    }

    if(index<2){        //Two landmarks at least are needed for the localiza-
tion of the robot
        std::cout<<"Not enough known landmarks for localization of the ro-
bot."<<std::endl;
        return;
```

```
    }

    else{
        int z=0;        //Getting all possible known landmarks combination

        int iterations=index*index;

        for(int i=index;i>0;i--){
            iterations=iterations-i;
        }

        float x[iterations],y[iterations], a[iterations], b[iterations];

        for(int i=0;i<index;i++){
            for(int j=0;j<index;j++){
                if(j>i){

                    Eigen::Matrix4f A;
                    A << -relativepos[i][1], relativepos[i][0], 1, 0,       //
-Y1r, X1r, 1, 0
                         relativepos[i][0], relativepos[i][1], 0, 1,        //
X1r, Y1r, 0, 1
                        -relativepos[j][1], relativepos[j][0], 1, 0,       //
-Y2r, X2r, 1, 0
                         relativepos[j][0], relativepos[j][1], 0, 1;       //
X2r, Y2r, 0, 1
                    Eigen::Vector4f c;
                    c<<known[i][0], known[i][1], known[j][0], known[j][1];
                    Eigen::Vector4f s = A.colPivHouseholderQr().solve(c);
//Solving with QR decomposition with column pivoting

                    a[z]=s[0];
                    b[z]=s[1];
                    x[z]=s[2];
                    y[z]=s[3];

                    z++;
                }
            }
        }

        float orientation=0;                    //Calculating average between
all landmarks combination
        float xtotal=0;
        float ytotal=0;

        for(int i=0;i<iterations;i++){

            //First we calculate orientation

            if(a[i]>1){     //Just in case some errors of sin or cos out of
range
                a[i]=1;
            }
            if(a[i]<-1){
                a[i]=-1;
            }
            if(b[i]>1){
                b[i]=1;
            }
            if(b[i]<-1){
                b[i]=-1;
            }

            if(a[i]>=0){
                orientation= orientation + acos(b[i]);
            }
            else{                               //If the sin is negative, we
are between 180 and 360, so the orientation is 360 - alpha(calculated from the
sin)
```

```
                orientation = orientation + 2*PI - acos(b[i]);
            }
            xtotal=xtotal+x[i];
            ytotal=ytotal+y[i];

        }

        robots[totalRobots].x=xtotal/iterations;
        robots[totalRobots].y=ytotal/iterations;
        robots[totalRobots].orientation=orientation/iterations;
        robots[totalRobots].name=name;

        totalRobots++;

    }

}



void Map::addunknownlandmarks(boost::shared_ptr<AL::ALProxy> nao, std::string
name){

    bool j=0;       //Check if the robot has been positioned already
    for(int i=0;i<totalRobots;i++){
        if(robots[i].name==name){
            j=1;
        }
    }
    if(j==0){
        return;
    }

    int c=0;
    for(int i=0;i<nao->call<int>("getLandmarksDetectedFixed");i++){
        for(int j=0;j<totalKnownLandmarks;j++){
            if(knownLandmarks[j].name==nao->call<AL::ALValue>("getMarkerID-
fixed", i).toString()){c++;}
        }
    }
    if(c==nao->call<int>("getLandmarksDetectedFixed")){return;}       //If all
the landrmarks detected by the robot are already known, there is no unknown
one (we suposse the robot see all the known landmarks)

    float relativepos[nao->call<int>("getLandmarksDetectedFixed")-totalKnown-
Landmarks][2];
    AL::ALValue id[nao->call<int>("getLandmarksDetectedFixed")-totalKnownLand-
marks];
    int index=0;

    for(int i=0; i<nao->call<int>("getLandmarksDetectedFixed");i++){
//Getting the data of the unknown landmarks detected by the robot
        int count=0;
        for(int j=0;j<totalKnownLandmarks;j++){
            if(knownLandmarks[j].name!=nao->call<AL::ALValue>("getMarkerID-
fixed", i).toString()){count++;}
            if(count==totalKnownLandmarks){
                id[index]= nao->call<AL::ALValue>("getMarkerIDfixed", i);
                relativepos[index][0]=nao->call<AL::ALValue>("getxav", i);
                relativepos[index][1]=nao->call<AL::ALValue>("getyav", i);
                index++;
            }
        }
    }

    int robotnumber;        //Searching for the number of the robot
    for(int i=0;i<totalRobots;i++){
        if(robots[i].name==name){
            robotnumber=i;
        }
```

```
    }

    for(int i=0;i<nao->call<int>("getLandmarksDetectedFixed")-totalKnownLand-
marks;i++){     //Calculating the position of the unknown landmarks
        unknownLandmarks[totalUnknownLandmarks].name=id[i];
        unknownLandmarks[totalUnknownLandmarks].x= relativepos[i][0] * cos(ro-
bots[robotnumber].orientation) - relativepos[i][1] * sin(robots[robotnum-
ber].orientation) + robots[robotnumber].x;      //Xl= Xlr * cos(alpha) - Ylr *
sin(alpha) + Xrobot
        unknownLandmarks[totalUnknownLandmarks].y= relativepos[i][0] * sin(ro-
bots[robotnumber].orientation) + relativepos[i][1] * cos(robots[robotnum-
ber].orientation) + robots[robotnumber].y;      //Yl= Xlr * sin(alpha) - Ylr *
cos(alpha) + Yrobot

        totalUnknownLandmarks++;

    }

}



void Map::exporttofile(std::string nameoffile){

    std::ofstream file;
    file.open(nameoffile.c_str());      //Columns separated by commas

    file << "Type,ID,X,Y,Orientation.\n";

    for(int i=0;i<totalRobots;i++){
        file << "Robot,"<< std::setprecision(2) << robots[i].name << ","<<ro-
bots[i].x << "," << robots[i].y << "," << robots[i].orientation << "\n";
    }

    for(int i=0;i<totalKnownLandmarks;i++){
        file << "KnownLandmark,"<< std::setprecision(2)<< knownLand-
marks[i].name << ","<<knownLandmarks[i].x << "," << knownLandmarks[i].y << ",-
\n";
    }

    for(int i=0;i<totalUnknownLandmarks;i++){
        file << "UnknownLandmark,"<< std::setprecision(2)<< unknownLand-
marks[i].name << ","<<unknownLandmarks[i].x << "," << unknownLandmarks[i].y <<
",-\n";
    }

    file.close();
}



void Map::saydistancetorobot(boost::shared_ptr<AL::ALProxy> nao, std::string
namefrom, std::string nameto){

    RobotInMap from, to;        //Searching for the robots

    for(int i=0;i<totalRobots;i++){
        if(robots[i].name==namefrom){
            from.name=robots[i].name;
            from.x=robots[i].x;
            from.y=robots[i].y;
        }
    }

    for(int j=0;j<totalRobots;j++){
        if(robots[j].name==nameto){
            to.name=robots[j].name;
            to.x=robots[j].x;
            to.y=robots[j].y;
        }
```

```
    }

    //Saying distance depending of the coordinates of each robot

    if(to.x>from.x){
        std::stringstream streamx;
        streamx << to.name << " is " <<std::setprecision(2)<< to.x-from.x << "
metres on my right, and ";
        nao->callVoid("say", streamx.str());
        std::cout<<streamx.str();
    }
    else{
        std::stringstream streamx;
        streamx << to.name << " is " <<std::setprecision(2)<< from.x-to.x << "
metres on my left, and ";
        nao->callVoid("say", streamx.str());
        std::cout<<streamx.str();
    }

    if(to.y>from.y){
        std::stringstream streamy;
        streamy <<std::setprecision(2)<< to.y-from.y << " metres on my
front."<<std::endl;
        nao->callVoid("say", streamy.str());
        std::cout<<streamy.str();
    }
    else{
        std::stringstream streamy;
        streamy  <<std::setprecision(2)<< from.y-to.y << " metres on my
back."<<std::endl;
        nao->callVoid("say", streamy.str());
        std::cout<<streamy.str();
    }

}


//Auxiliary methods to display the robots, knownlandmarks and unknownlandmarks

void Map::showrobots(){
    qiLogInfo("Map") << "Robots in the map: "<< robots[0].name << " "<< ro-
bots[1].name << " "<< robots[2].name << " "<< robots[3].name << " "<< ro-
bots[4].name <<
                        " . Orientation of the robots: " << robots[0].orienta-
tion << " "<< robots[1].orientation << " "<< robots[2].orientation << " "<<
robots[3].orientation << " "<< robots[4].orientation <<
                    ". X of the robots: " << robots[0].x << " "<< robots[1].x
<< " "<< robots[2].x << " "<< robots[3].x << " "<< robots[4].x <<
                        ". Y of the robots: " << robots[0].y << " "<< ro-
bots[1].y << " "<< robots[2].y << " "<< robots[3].y << " "<< robots[4].y <<
std::endl;
}



void Map::showknownlandmarks(){
    qiLogInfo("Map") << "Knownlandmarks in the map: "<< knownLandmarks[0].name
<< " "<< knownLandmarks[1].name << " "<< knownLandmarks[2].name << " "<<
knownLandmarks[3].name << " "<< knownLandmarks[4].name <<
                    ". X of the knownlandmarks: " << knownLandmarks[0].x << "
"<< knownLandmarks[1].x << " "<< knownLandmarks[2].x << " "<< knownLand-
marks[3].x << " "<< knownLandmarks[4].x <<
                        ". Y of the knownlandmarks: " << knownLandmarks[0].y
<< " "<< knownLandmarks[1].y << " "<< knownLandmarks[2].y << " "<< knownLand-
marks[3].y << " "<< knownLandmarks[4].y << std::endl;

}
```

```
void Map::showunknownlandmarks(){

    qiLogInfo("Map") << "Unknownlandmarks in the map: "<< unknownLand-
marks[0].name << " "<< unknownLandmarks[1].name << " "<< unknownLand-
marks[2].name << " "<< unknownLandmarks[3].name << " "<< unknownLand-
marks[4].name <<
                        unknownLandmarks[5].name << " "<< unknownLand-
marks[6].name << " "<< unknownLandmarks[7].name << " "<< unknownLand-
marks[8].name << " "<< unknownLandmarks[9].name <<
                        unknownLandmarks[10].name << " "<< unknownLand-
marks[11].name << " "<< unknownLandmarks[12].name << " "<< unknownLand-
marks[13].name << " "<< unknownLandmarks[14].name <<
                        unknownLandmarks[15].name << " "<< unknownLand-
marks[16].name << " "<< unknownLandmarks[17].name << " "<< unknownLand-
marks[18].name << " "<< unknownLandmarks[19].name << std::endl <<
                    ". X of the ununknownLandmarks: " << unknownLandmarks[0].x
<< " "<< unknownLandmarks[1].x << " "<< unknownLandmarks[2].x << " "<< un-
knownLandmarks[3].x << " "<< unknownLandmarks[4].x << " "<<
                        unknownLandmarks[5].x << " "<< unknownLandmarks[6].x
<< " "<< unknownLandmarks[7].x << " "<< unknownLandmarks[8].x << " "<< un-
knownLandmarks[9].x << " " <<
                        unknownLandmarks[10].x << " "<< unknownLand-
marks[11].x << " "<< unknownLandmarks[12].x << " "<< unknownLandmarks[13].x <<
" "<< unknownLandmarks[14].x << " " <<
                        unknownLandmarks[15].x << " "<< unknownLand-
marks[16].x << " "<< unknownLandmarks[17].x << " "<< unknownLandmarks[18].x <<
" "<< unknownLandmarks[19].x << std::endl <<
                        ". Y of the ununknownLandmarks: " << unknownLand-
marks[0].y << " "<< unknownLandmarks[1].y << " "<< unknownLandmarks[2].y << "
"<< unknownLandmarks[3].y << " "<< unknownLandmarks[4].y << " " <<
                        unknownLandmarks[5].y << " "<< unknownLandmarks[6].y
<< " "<< unknownLandmarks[7].y << " "<< unknownLandmarks[8].y << " "<< un-
knownLandmarks[9].y << " " <<
                        unknownLandmarks[10].y << " "<< unknownLandmarks[11].y
<< " "<< unknownLandmarks[12].y << " "<< unknownLandmarks[13].y << " "<< un-
knownLandmarks[14].y << " "<<
                        unknownLandmarks[15].y << " "<< unknownLandmarks[16].y
<< " "<< unknownLandmarks[17].y << " "<< unknownLandmarks[18].y << " "<< un-
knownLandmarks[19].y << std::endl;
}
```

MAIN

```
#include <signal.h>
#include <boost/shared_ptr.hpp>
#include <alcommon/albroker.h>
#include <alcommon/almodule.h>
#include <alcommon/albrokermanager.h>
#include <alcommon/altoolsmain.h>
#include <pthread.h>

#include "map.h"
#include "robot.h"

#define BLUEIP "192.168.1.59"
#define ORANGEIP "192.168.1.34"
#define PORT 9559
#define BROKERIP "0.0.0.1"



int main(int argc, char *argv[])
{
    int brokerPort = 54000;

    boost::shared_ptr<AL::ALBroker> bluebroker= AL::ALBroker::cre-
ateBroker("bluebroker", BROKERIP, brokerPort, BLUEIP, PORT, 0);
```

```
    boost::shared_ptr<AL::ALBroker> orangebroker = AL::ALBroker::createBroker(
"orangebroker", BROKERIP, brokerPort, ORANGEIP, PORT, 0);

    AL::ALBrokerManager::setInstance(bluebroker->fBrokerManager.lock());
    AL::ALBrokerManager::getInstance()->addBroker(bluebroker);

    AL::ALBrokerManager::setInstance(orangebroker->fBrokerManager.lock());
    AL::ALBrokerManager::getInstance()->addBroker(orangebroker);


    *AL::ALModule::createModule<Robot>(bluebroker, "Robot");

    boost::shared_ptr<AL::ALProxy> bluenao =
boost::shared_ptr<AL::ALProxy>(new AL::ALProxy(bluebroker, "Robot"));

    bluenao->callVoid("configure");

    bluenao->callVoid("scan");

    bluenao->callVoid("calculatepositions");

    bluenao->callVoid("fixpositions");

    showAveragedPositiondsDetected(bluenao, "bluenao");

    bluenao->callVoid("exporttofile", "demoblue.csv");


    AL::ALModule::createModule<Robot>(orangebroker, "Robot");

    boost::shared_ptr<AL::ALProxy> orangenao =
boost::shared_ptr<AL::ALProxy>(new AL::ALProxy(orangebroker, "Robot"));

    orangenao->callVoid("configure");

    orangenao->callVoid("scan");

    orangenao->callVoid("calculatepositions");

    orangenao->callVoid("fixpositions");

    showAveragedPositiondsDetected(orangenao, "orangenao");

    orangenao->callVoid("exporttofile", "demoorange.csv");


    Map mymap;

    mymap.initialize();

    mymap.addknownlandmark("[68]", 1.11f, 2.4f);

    mymap.addknownlandmark("[119]", 1.6f, 1.89f);

    mymap.addknownlandmark("[80]", 0.75, 2.15f);

    mymap.addrobot(bluenao, "bluenao");

    mymap.addrobot(orangenao, "orangenao");

    mymap.addunknownlandmarks(bluenao, "bluenao");

    mymap.addunknownlandmarks(orangenao, "orangenao");

    mymap.exporttofile("demo.csv");

    mymap.saydistancetorobot(bluenao, "bluenao", "orangenao");
}
```